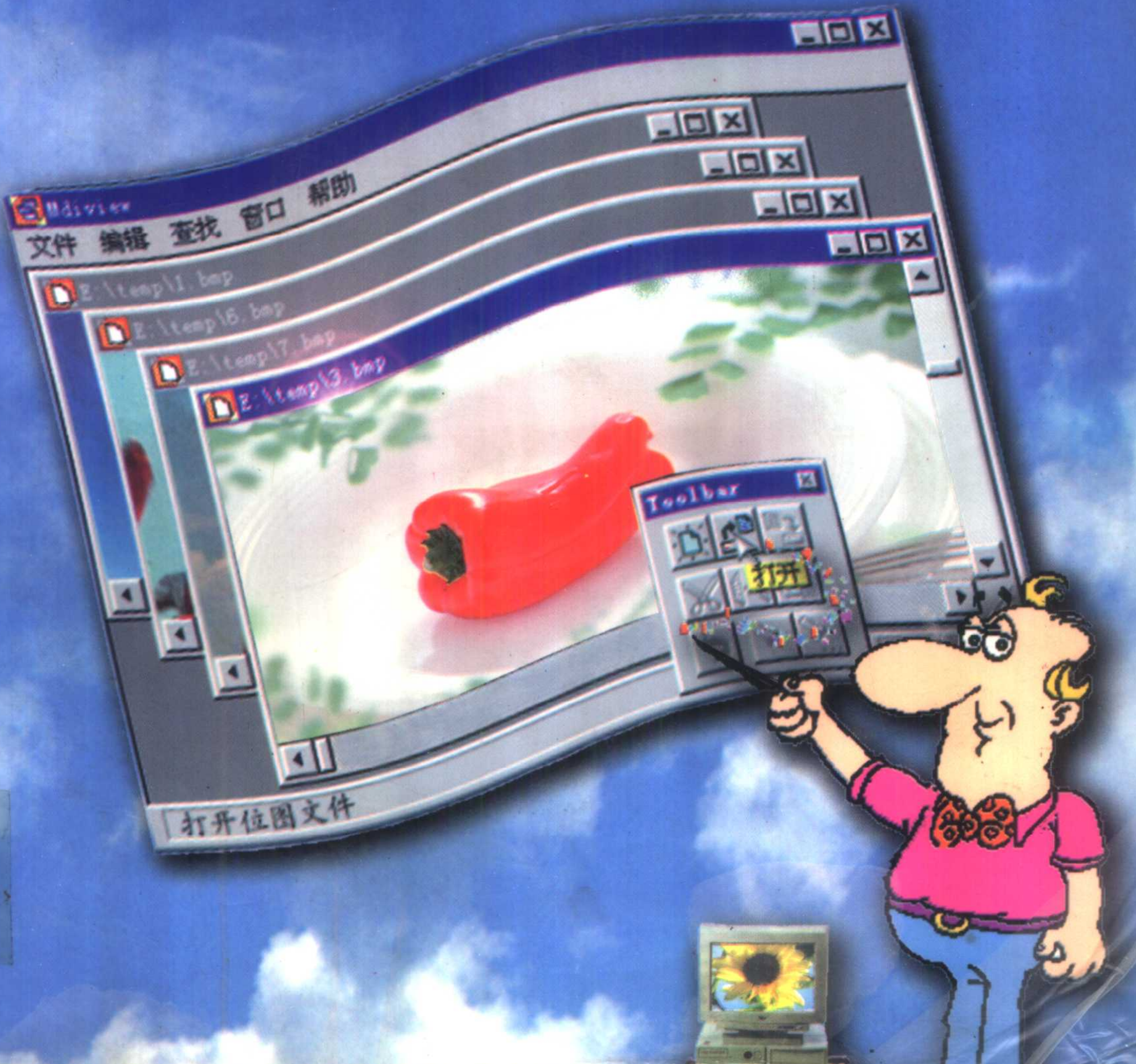
Borland C++ 5.0

OWL 5.0 编程技术与实例

雷宇 江明霞 代国定 高宏福



西安电子科技大学出版社

Borland C++ 5.0

OWL 5.0 编程技术与实例

雷 宇 江明霞 代国定 高宏福

西安电子科技大学出版社

1997

（陕）新登字 010 号

内 容 简 介

本书介绍了 Borland C++ 5.0 中的面向对象类库 OWL 5.0 及其编程技术。OWL 是支持高效编写 Windows 95 与 Window NT 的 32 位应用程序的有力工具。本书中包括了 Windows 95 程序设计基本概念、OWL 基础、各种控件类、对话框类、单文档和多文档界面及 OWL 的类库说明。针对 OWL 中的主要控件，书中给出了大量的程序实例代码。另外，本书通过一个较大、较系统的实例由浅入深地介绍了 OWL 应用程序的开发过程及开发技巧。这些内容对于学习面向对象的程序设计技术及学习 Windows 95 程序开发技术都会有较大的帮助。

本书可作为初学面向对象编程者的自学教材，更可作为想深入了解和掌握 OWL 编程技术的各类读者的参考书。

Borland C++ 5.0 OWL 5.0 编程技术与实例

雷 宇 江明霞 代国定 高宏福 编著

责任编辑 陈宇光

西安电子科技大学出版社出版发行

地址：西安市太白南路 2 号 邮编：710071

西安市长青印刷厂印刷

各地新华书店经销

开本 787×1092 1/16 印张 27 字数 645 千字

1997 年 6 月第 1 版 1997 年 6 月第 1 次印刷 印数 1—6 000

ISBN 7-5606-0523-0/TP·0255 定价：35.50 元

序言　用OWL 5.0编程

1995年到1996年期间，信息技术的发展真是令人兴奋。几乎所有的计算机杂志和报纸都在讨论未来的桌面计算机的发展趋势，讨论Internet的发展，讨论Intranet的结构，两家大公司为了争夺个人电脑32位操作系统的统制权在进行着兵团作战，各种各样的应用软件也在真刀真枪的拼杀；多少公司被收购，多少公司脱颖而出。

时至今日，有一点已经非常明朗：对于商业软件、家用软件和数据库前端软件的开发商和广大用户来说，Windows 95或Windows NT是最合适的32位桌面操作系统。尤其是Windows 95，它采用了新的用户界面，更加易学、易用，系统更加稳定、高效。

都知道Windows 95好用，但Windows 95程序难编。Windows 95是一个复杂的系统，它对应用程序有严格的要求，使得通常的程序设计方法难度大而效率低。为了摆脱这个局面，许多软件公司推出了基于Windows 95的程序开发工具，其中有Microsoft公司的Visual Basic 4、Visual C++ 4，Powersoft公司的Powerbuilder 5，Borland公司的Delphi 2、BC++ 5.0、Intrabuilder等等，它们都能不同程度的满足用户的需要，而且各自都有许多过人之处，用户可以针对不同的任务选择不同的开发工具。

那么在什么情况下选择BC++ 5.0作为Windows 95的开发工具呢？

1. 如果希望尽量发挥Windows 95系统的功能并且展示你自己的风格，还想要有一个高效率，希望利用BC++ 5.0所提供的工具尽量减少手工编制代码的繁琐过程；

2. 如果需要做大量的科学计算(包括图形、图像处理)，并且充分利用以前在BC上所做的程序代码；

3. 如果希望同时开发基于Windows不同版本的应用程序(比如基于Windows 3.x的16位应用程序和基于Windows 95或Windows NT的32位应用程序)，BC++ 5.0是能够满足这一要求的为数不多的开发工具，BC++ 5.0甚至依然能够开发高效的DOS应用程序，这是Borland公司为方便用户而努力的结果；

4. 如果希望快速开发“专业”的Windows 95应用程序，充分发挥面向对象和可视化编程的特点，那么BC++ 5.0是合适的选择，因为它包含了灵活而高效的OWL 5.0。

OWL不是猫头鹰，OWL 5.0(Object Windows Library)是BC++ 5.0中的精华，它采用面向对象技术封装了Windows的应用程序接口，使C++程序员能够高效地编写Windows程序。它提供了Windows中的界面元素(比如，窗口、按钮等)的“类”形式，使程序员更容易控制程序的用户界面；它提供了更为简洁的事件驱动定义方式，摆脱了用C语言时的繁琐模式。

OWL从BC++ 3.1中的OWL 1.0发展到BC++ 5中的OWL 5.0，其间变化剧烈，把面向对象的Windows编程技术一步步提高到新的水平，因而受到软件开发商和越来越多的C++程序员的青睐。

凡是用过OWL的C++用户都会有同样的感受：用OWL开发Windows程序要比传统的开发方式优越得多，它省去了不少编程时的麻烦，把程序员从繁重的近似于“体力劳动”

的工作中解脱出来，而能把主要的精力放在软件的功能上。BC++ 5.0 面市以来，OWL 的优势又一次将 Windows 编程方式提高到新的水平。OWL 5.0 几乎覆盖所有 16 位和 32 位的 Windows API 函数，它的封装自然而富有逻辑，总让人觉得 Windows 的面向对象机制本来就是这样的。OWL 5.0 中包括的一系列高层次的类，比如，工具条、状态条、浮动窗口、打印预览以及菜单类等等，都可以帮助你建立起复杂的应用程序。

对于想同时进行 16 位和 32 位 Windows 应用程序开发的人员来说，OWL 5.0 是再合适不过的了。只要利用多目标项目管理器、TargetExpert 和 Source Pools 来管理目标程序，就可以轻而易举地将相同的程序原代码在 16 位和 32 位 Windows 目标代码之间切换。

如果想学习 OWL，而你又没有多少 C++的经验，不要紧，学习 OWL 本身就可以是学习 C++的一个精彩的开头。OWL 中的类结构的规范性和灵活性是你在任何 C++教程中所不常见到的；如果想学习 Windows 编程，OWL 中的类层次结构和体系可以帮助你正确的理解 Windows 编程中的难懂的概念。它就是你编写 Windows 程序的开始。OWL 就是 Borland 公司用 C++语言强化了 Windows 中的面向对象机制的结果，所以在你学习 OWL 的过程中，就可以深入地理解 C++的用法和 Windows 的编程方法。

在这本书中包括了 OWL 5.0 中常用的和有用的类的详细解释，尤其对编程实践中经常遇到的类给出了大量的程序实例，书中还包括了 OWL 的一些使用技巧，可以使读者更加深入地理解 OWL 的用法。

衷心感谢杨银堂老师、周端老师对这本书的大力支持！衷心感谢陈宇光老师在本书出版过程中的大力帮助！

作者水平有限，书中难免存在错误和不足之处，敬请批评指正。

衷心希望本书会对您的事业有些帮助。

编　者

目　录

第一章　Windows 95 程序设计基本概念 …… 1
1.1　Windows 95 的优越性 …… 1
1.2　窗口的组成 …… 1
1.3　32 位 Window 应用程序接口与 OWL 5.0 …… 2
1.4　消息循环与事件驱动 …… 2
1.5　窗口过程 …… 3
1.6　C 或 CPP 文件 …… 3
1.7　资源与资源文件 …… 3
1.8　模块定义文件 …… 3
1.9　最基本的 Window 应用程序 …… 4
第二章　OWL 5.0 基础 …… 6
2.1　窗口和 TWindow 类 …… 6
2.2　从最简单的程序开始 …… 9
2.3　应用程序和 TApplication 类 …… 9
2.4　具有主窗口的应用程序 …… 11
2.5　加入客户窗口 …… 11
2.6　让窗口响应事件 …… 13
2.7　资源与资源编辑器 …… 15
2.8　事件响应程序实例 …… 17
2.9　AppExpert 和 ClassExpert …… 24
2.10　AppExpert 程序实例 …… 24
第三章　控件类 …… 31
3.1　什么是控件对象 …… 31
3.2　按钮控件 …… 33
3.3　检查框控件 …… 34
3.4　Radio 按钮控件 …… 36
3.5　组框控件 …… 37
3.6　按钮控件程序实例(一) …… 37
3.7　按钮控件程序实例(二) …… 41
3.8　静态控件 …… 45
3.9　编辑框控件 …… 46
3.10　编辑框控件程序实例 …… 47
3.11　列表框控件 …… 54
3.12　列表框控件程序实例 …… 55
3.13　组合框控件 …… 59
3.14　组合框控件程序实例 …… 61
3.15　滚动条控件 …… 65
3.16　滚动条控件程序实例 …… 66
3.17　Slider 控件 …… 71
3.18　Gauge 控件 …… 72
3.19　Gauge 控件程序实例 …… 73
第四章　对话框类 …… 77
4.1　对话框 …… 77
4.2　各种风格的对话框 …… 77
4.3　构造对话框对象 …… 78
4.4　执行一个对话框对象 …… 79
4.5　关闭对话框 …… 81
4.6　在资源编辑器中编辑对话框 …… 81
4.7　用对话框作主窗口 …… 81
4.8　界面对象与控件的联系 …… 82
4.9　管理对话框 …… 83
4.10　使用传输缓冲区 …… 84
4.11　模式对话框程序实例 …… 86
4.12　无模式对话框程序实例 …… 96
4.13　传输缓冲区程序实例 …… 97
4.14　输入框 …… 99
4.15　打印中断框 …… 100
4.16　通用对话框 …… 100
4.17　颜色选择对话框 …… 101
4.18　查找和替换对话框 …… 102
4.19　字体选择对话框 …… 104
4.20　打印对话框 …… 106
4.21　文件打开和保存对话框 …… 108
4.22　通用对话框程序实例 …… 111
4.23　多页对话框程序实例 …… 128
4.24　信息框 …… 135
第五章　OWL 5.0 应用实例 …… 137
5.1　编一个有趣的程序“生物圈三号” …… 137
5.2　将“生物圈三号”改造成一个屏幕保护程序 …… 193
第六章　单文档界面和多文档界面 …… 221
6.1　单文档位图观察器 …… 221

6.2 多文档位图观察器 ························ 234

附录 A ObjectWindows 5.0 类库 ········ 249

A.1 TWindow 类 ······························ 249
A.2 TWindowAttr 结构 ······················ 265
A.3 TApplication 类 ························ 267
A.4 TFrameWindow 类 ······················ 271
A.5 TControl 类 ···························· 274
A.6 TScrollBar 类 ·························· 276
A.7 TScrollBarData 结构 ···················· 278
A.8 TSlider 类 ····························· 278
A.9 THSlider 类 ···························· 282
A.10 TVSlider 类 ··························· 283
A.11 TGauge 类 ···························· 283
A.12 TGroupBox 类 ························· 285
A.13 TStatic 类 ···························· 286
A.14 TEdit 类 ····························· 287
A.15 TEditSearch 类 ························ 291
A.16 TEditFile 类 ·························· 293
A.17 TButton 类 ··························· 294
A.18 TCheckBox 类 ························· 295
A.19 TRadioButton 类 ······················ 297
A.20 TVbxControl 类 ······················· 298
A.21 TListBox 类 ·························· 301
A.22 TListBoxData 结构 ···················· 304
A.23 TComboBox 类 ························ 305
A.24 TComboBoxData 结构 ·················· 307
A.25 TValidator 类 ························· 309
A.26 TFilterValidator 类 ···················· 310
A.27 TRangeValidator 类 ···················· 310
A.28 TPXPictureValidator 类 ················ 311
A.29 TLookupVahdator 类 ··················· 311
A.30 TStringLookupValidator 类 ············· 312
A.31 TDialog 类 ··························· 312
A.32 TInputDialog 类 ······················ 315
A.33 TPninterAbortDlg 类 ·················· 315
A.34 TCommonDialog 类 ···················· 316
A.35 TChoosColorDialog 类 ················· 317
A.36 TChooseColorDialog::TData 类 ········ 318
A.37 TFindReplaceDialog 类 ················ 319
A.38 TFindReplaoeDialog::TData 类 ········ 320
A.39 TFindDialog 类 ······················· 321
A.40 TReplaceDialog 类 ···················· 322
A.41 TChooseFontDialog 类 ················· 322
A.42 TChooseFontDialog::TData 类 ········· 323
A.43 TPrinterDialog 类 ····················· 325
A.44 TOpenSaveDialog 类 ··················· 325
A.45 TOpenSaveDialog::TData 类 ··········· 326
A.46 TFileOpenDialog 类 ···················· 328
A.47 TFileSaveDialog 类 ···················· 328
A.48 TGdiObject 类 ························ 329
A.49 TRegion 类 ··························· 331
A.50 TBitmap 类 ··························· 332
A.51 TFont 类 ····························· 334
A.52 TGadgetWindowFont 类 ················ 335
A.53 TPalette 类 ··························· 336
A.54 TBrush 类 ···························· 338
A.55 TPen 类 ····························· 338
A.56 TIcon 类 ····························· 340
A.57 TCursor 害 ··························· 340
A.58 TDib 类 ····························· 341
A.59 TDC 类 ······························ 345
A.60 TWindowDC 类 ························ 368
A.61 TDesktopDC 类 ························ 368
A.62 TScreenDC 类 ························· 368
A.63 TClientDC 类 ························· 369
A.64 TOleClient 类 ························ 369
A.65 TPaintDC 类 ·························· 369
A.66 TMetaFileDC 类 ······················· 369
A.67 TCreatedID 类 ························ 370
A.68 TDibDC 类 ··························· 370
A.69 TPrintDC 类 ·························· 370
A.70 TIC 类 ······························ 372
A.71 TMemoryDC 类 ························ 372
A.72 TColor 类 ···························· 372
A.73 TLayoutWindow 类 ···················· 374
A.74 TLayoutConstraint 结构 ················ 375
A.75 TLayoutMetrics 类 ···················· 376
A.76 TMDIClient 类 ························ 377
A.77 TMDIFrame 类 ························ 379
A.78 TMDIChild 类 ························ 380
A.79 TDecoratedFrame 类 ·················· 381
A.80 TDecoratedMDIFrame 类 ··············· 383
A.81 TGadgetWindow 类 ···················· 383
A.82 TToolBox 类 ·························· 387
A.83 TControlBar 类 ······················· 388
A.84 TMessageBar 类 ······················ 388

A. 85 TStatusBar 类 …… 389
A. 86 TGadget 类 …… 391
A. 87 TBitmapGadget 类 …… 394
A. 88 TButtonGadget 类 …… 395
A. 89 TTextGadget 类 …… 397
A. 90 TSeperatorGadget 类 …… 398
A. 91 TControlGadget 类 …… 399
A. 92 TMenu 类 …… 400
A. 93 TSystemMenu 类 …… 402
A. 94 TPopupMenu 类 …… 403
A. 95 TMenuDescr 类 …… 403
附录 B OWL 中的事件响应宏定义 …… 405
B. 1 事件响应机制 …… 405
B. 2 声明事件响应表 …… 405
B. 3 定义事件响应表 …… 405
B. 4 事件响应的宏 …… 406
附录 C OWL 中的常数定义 …… 415
C. 1 BF_xxxx 按按钮标志常数 …… 415
C. 2 CM_xxxx 编辑命令常数 …… 416
C. 3 CM_xxxx 编辑替换命令常数 …… 416
C. 4 CM_xxxx 编辑替换命令常数 …… 416
C. 5 CM_xxxx 多文档界面命令常数 …… 416
C. 6 dmxxxx 文档管理器模式常数 …… 417
C. 7 dtxxxx 文档样板常数 …… 417
C. 8 ID_xxxx 文件常数 …… 418
C. 9 ID_xxxx 打印常数 …… 419
C. 10 IDA_EDITFILE 加速键 ID 常数 …… 419
C. 11 IDA_OLEVIEW OLE 加速键 ID 常数 …… 419
C. 12 IDM_EDITFILE 菜单 ID 常数 …… 419
C. 13 IDS_xxxx 文档字符串 ID 常数 …… 419
C. 14 IDS_xxxx 编辑文件 ID 常数 …… 420
C. 15 IDS_xxxx 异常管理常数 …… 420
C. 16 IDS_xxxx 列表显示 ID 常数 …… 422
C. 17 IDS_xxxx 打印字符串常数 …… 422
C. 18 IDS_xxxx validator 常数 …… 422
C. 19 IDW_MDICLIENT 常数 …… 423
C. 20 IDW_FIRSTMDICHILD 常数 …… 423
C. 21 ImParent 常数 …… 423
C. 22 MAX_RSRC_ERROR_STRING 常数 …… 423
C. 23 pfxxxx 性能属性常数 …… 423
C. 24 vnxxxx 视口通知常数 …… 424
C. 25 TValidator Options 联合 …… 424

第 1 章
Windows 95 程序设计基本概念

1.1 Windows 95 的优越性

一般认为，Microsoft Windows 是一个优于 DOS 的图形用户界面的操作系统，它为应用程序提供了一致的窗口用户界面和多任务环境。而 32 位的 Windows 95 是比 16 位的 Windows 3.x 更具有优势的操作系统，它的新型用户界面更加容易学习和使用，32 位应用程序的执行速度更快，更能够充分发挥计算机硬件的功能，系统也更加稳定(相对于 Windows 3.x)，因此，目前 Windows 95 成为主流的桌面操作系统是必然的。

Windows 95 的界面对用户是友好的，Windows 95 的编程环境对程序员也是友好的。如果你有编写 DOS 应用程序的经验，你会记得给应用程序增加友好的界面是一项复杂的工作，所有的图形界面元素都需要从头编起。而在 Windows 95 环境中，一致的界面带来了一致的界面设计过程，通用的界面元素都已经建立(包括丰富的 OCX 和 VBX 控件)，你只需要学会使用它们，就可以在不同的程序中利用这些界面元素生成应用程序界面。你的主要精力可集中在编写应用程序的功能上。

因为是图形用户界面，所以在 Windows 95 环境中开发图形软件就比较方便；Windows 95 环境直接支持鼠标，所以在程序中使用鼠标功能是非常方便的；Windows 95 环境对内存的管理能力强，因而应用程序可以使用更多的内存资源；Windows 95 提供了“与设备无关”的程序运行环境，因此程序会不依赖于某一种显式硬件或打印机；Windows 95 中内含丰富的多媒体功能，使开发多媒体应用程序更加方便。

Windows 95 的优势一言难尽，相信它的用户会有体会。

1.2 窗口的组成

窗口是 Windows 95 程序进行屏幕输出的方式，它包括了一系列部件(见图1-1)，其中有：

窗口边框：窗口边框包围着窗口，它可以使窗口的大小改变；

客户区域：窗口中除了菜单条、滚动条、边框等窗口的外围部分以外的窗口的实际工作区域，窗口的客户区域由应用程序控制；

菜单条：菜单条是应用程序提供给用户的命令选择工具，菜单条由程序管理；

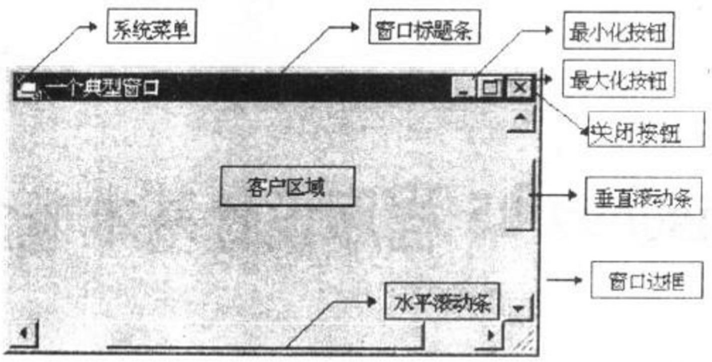


图 1-1　典型的 Windows 95 窗口

滚动条：包括水平滚动条和垂直滚动条，它可以帮助窗口显示比窗口实际尺寸大的文档或图形；

最大化按钮：用户点击最大化按钮可以使窗口最大化，充满整个屏幕；

最小化按钮：用户点击最小化按钮可以使窗口最小化，缩小为一个图标；

系统菜单：系统菜单在应用程序窗口的左上角，用户可以通过此菜单对系统进行操作，双击此处则可以关闭应用程序；

标题条：标题条显示出窗口的名称或应用程序的名称；

关闭程序按钮：用户点击此按钮可以关闭应用程序。

1.3　32 位 Windows 应用程序接口与 OWL 5.0

Windows 95 应用程序接口(Win32 API)函数是 Windows 95 应用程序的核心。通过 Win32 API 函数可以利用 Windows 95 的功能和“与设备无关性”，并可编制出运行于 Windows 95 与 WindowsNT 的 32 位应用程序。不过，Win32 API 中的函数达数千个之多，这足以使程序员们感到头昏眼花。OWL 5.0 正是为这些痛苦不堪的程序员们提供的妙方，它采用清晰的逻辑、自然的结构，用面向对象的方法封装了几乎所有的函数，使用亦非常方便。

BC++ 5.0 中包含了一个 Microsoft 公司编写的专门用于 Windows 环境的游戏开发应用程序接口(WinG)，它使游戏程序可以像在 DOS 环境中一样快速显示图形，这对那些热心 Windows 95 环境下开发游戏程序的用户来讲真是一个福音。

另外，WindowsNT 中包含了从 SGI 工作站移植过来的 3D 图形库 OpenGL，它使高性能的 WindowsNT 的工作站可以像图形工作站那样进行复杂的图形处理。BC++ 5.0 中提供了 OpenGL 的应用程序接口，这对于 CAD 或图形处理的用户又是一个福音。

1.4　消息循环与事件驱动

DOS 程序一般采用顺序的、过程驱动的结构，而 Windows 95 程序是采用事件驱动的结构方式(见图 1-2)。事件驱动程序不是按某种固定顺序来执行的，而是由事件的发生来控制执行的。通常，事件以消息的方式进入程序，比如，当鼠标的一个按钮按下时，应用程序将接到一个鼠标按钮按下的消息，程序执行这个消息的响应程序。

事件驱动的程序在启动初始化后就进入了消息循环中。Windows 95 应用程序接收的消息种类很多，包括：输入设备的消息、键盘的状态、设备的状态和时钟消息等。Windows 95 负责把所有输入设备的消息放入系统消息队列中，然后将消息分配到各个应用程序的消息队列中，由应用程序处理。

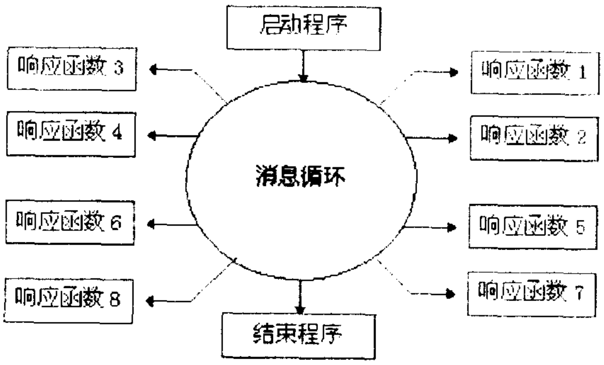


图 1 - 2 Windows 95 程序的事件驱动方式

1.5 窗口过程

窗口过程是接收和处理消息的函数，它负责执行不同的代码来响应不同的消息。在 C 语言中，窗口过程利用 switch 语句进行消息响应，因而比较繁琐。使用 OWL 5.0 可以大大简化这个编程过程。

1.6 C 或 CPP 文件

以 C 或 CPP 为扩展名的文件包含了 Windows 95 应用程序的源代码，在项目文件中可以有多个 C 或 CPP 文件。

1.7 资源与资源文件

一般的 Windows 95 应用程序都需要使用图标、菜单、光标、位图、对话框之类的资源，资源文件（*.rc）就是用来存放这些资源的。项目文件中可以有多个资源文件。在应用程序的开发中，可以使用 Resource Workshop（资源编辑器）来开发应用程序的资源，资源编辑器将资源文件与编译好的可执行模块 EXE 或 DLL 连接成为完整的 Windows 95 应用程序。

1.8 模块定义文件

定义文件（DEF）中提供了应用程序的特别重要的信息，包括：应用程序性质、代码和数据段大小和性质、局部堆的大小等，如果在项目文件中不包括这个文件，Borland C++ 会使用一个缺省的定义文件。

1.9 最基本的 Windows 应用程序

下面是一个用 C 语言编写的最基本的 Windows 应用程序，它可以改变窗口的大小、能够移动、关闭、最小化、最大化窗口。

```
/*******************************************
        最基本的 Windows 应用程序
*******************************************/
#include <Windows.h>
#pragma hdrstop

LRESULT FAR PASCAL export WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)

//Windows 应用程序的主窗口
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpszCmd, int nCmdShow)
{
  HWND         hwnd ;
  MSG          msg ;
  WNDCLASS     wndclass ;
  //设置窗口类，登记一个窗口
  if (! hPrevInstance)
  {
     wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc    = (WNDPROC)WndProc ;
     wndclass.cbClsExtra     = 0 ;
     wndclass.cbWndExtra     = 0 ;
     wndclass.hInstance      = hInstance ;
     wndclass.hIcon          = LoadIcon( NULL, IDI_APPLICATION );
     wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground  = (HBRUSH)(COLOR_WINDOW + 1);
     wndclass.lpszMenuName   = MAKEINTRESOURCE( CHELPAPMENU );
     wndclass.lpszClassName  = szAppName ;

     RegisterClass (&wndclass) ;
  }

  //创建一个窗口
  hwnd = CreateWindow (szAppName,
    "Windows 95 Help Example",
    WS_OVERLAPPEDWINDOW,
```

```
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL) ;

    //显示窗口
    ShowWindow (hwnd, nCmdShow) ;
    UpdateWindow (hwnd) ;

    //消息循环
    while (GetMessage (&msg, NULL, 0, 0))
    {
      if ( ! TranslateAccelerator( hwnd, hAccel, &msg ))
      {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
      }
    }
    return msg. wParam ;
}

//窗口过程
LRESULT FAR PASCAL export WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
    case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

第 2 章

OWL 5.0 基础

2.1　窗口和 TWindow 类

Windows 的核心是窗口。前面已经介绍了窗口的基本概念，在 OWL 5.0 中，TWindow 是所有窗口类的基类，它封装了 Windows 中窗口的基本特性和基本功能。

TWindow 类所封装的窗口特性和与窗口有关的一系列成员函数，提供了窗口的初始化、创建、操作及关闭窗口的功能。可以直接用 TWindow 来构造一个窗口对象，但是通常不这样做，往往只把它作为一个更有个性的窗口类的基类。

TWindow 派生出一系列窗口类的子类，包括 TFrameWindow，TControl，TDialog 和 TMDIChild 类等。从 ObjectWindows 的类结构层次图可以看到 TWindow 类与它的一部分子类的关系（见图 2－1）。

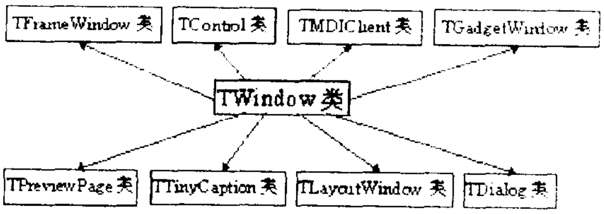


图 2－1　TWindow 类与它的派生类

如何用一个窗口类实现一个典型的窗口？我们可以用下面的两个步骤完成：

1. 用 TWindow 类或者它的派生类构造一个窗口对象；

2. 调用 TWindow 类中的成员函数 Create 或 Execute 来创建窗口界面元素（窗口的实际显示部分）。

例如：

```
TWindow * window = new TWindow(this);
window->Create();
```

如果使用 TWindow 类的派生类（比如 TNewWindow）创建窗口也是同样的。

例如：

```
TNewWindow * window = new TNewWindow(this);
window->Create();
```

TWindow::Create()函数为窗口的创建做了一些必要的准备工作，包括：

1. 创建窗口界面元素；
2. 设置窗口句柄 HWindow ；
3. 设置窗口属性参数(包括窗口大小、显示方式等)；
4. 调用 TWindow::SetupWindow()。

在调用 Create 函数时你当然可以不必关心这个函数内部的事情，但知道这个“秘密”是有好处的！你可以在调用 Create 函数创建窗口前设置窗口的属性 TWindow::Attr(窗口类的数据成员)，以得到想要的窗口风格；你还可以在 TWindow 的派生类中重载 SetupWindow()函数(这是窗口创建前调用的最后一个函数)，并用它做一些窗口的初始化工作。之后，这个窗口就可以交付使用了。

当程序关闭这个窗口时，根据具体情况不同，用下面的两个方法之一来删除窗口界面元素(与 Create 函数对应)：

1. 调用成员函数 Destroy 来删除窗口界面元素；
2. 调用成员函数 CloseWindow，它又会调用成员函数 CanClose 弹出一个对话框让用户确认这一操作，如果用户确认就删除窗口界面元素。

当窗口对象不再使用时，有两个方法来删除它(与通常的对象删除方法完全一致)：

1. 如果窗口对象是 new 创建的，使用 delete 删除；
2. 如果窗口对象不是 new 创建的，它会被自动删除。

一个典型的 Windows 应用程序可以具有许多不同风格的窗口对象：有的是覆盖窗口、有的是弹出窗口，有的是有边界窗口，有的是有滚动条窗口，还有的是有标题窗口。不同的窗口风格是通过设置窗口属性参数 TWindowAttr Attr 确定的。窗口风格和窗口标题都是在窗口初始化时设置的，创建窗口元素时将使用这些参数。

一个窗口对象的属性参数包括窗口风格和窗口标题等一系列内容，表 2-1 是窗口属性类型的数据成员。

表 2-1

成员	类型	描　　述
Style	uint32	风格内容
ExStyle	uint32	扩展风格内容
X	int	窗口左上角在屏幕的横坐标
Y	int	窗口左上角在屏幕的纵坐标
W	int	窗口的初始化宽度
H	int	窗口的初始化高度
Menu	TResId	窗口菜单的 ID 值
Id	int	子窗口的 ID 值
Param	char far *	被 TMDIClient 用作保存多文档界面信息
AccelTable	TResId	窗口的加速键表 ID 值

下面的程序代码用 TWindow 类构造了一个窗口对象(见图 2-2)，并创建了与这个窗口对象相对应的窗口界面元素。然后设置了窗口的属性，显示这个窗口，最后删除界面元素和窗口对象。这就是窗口短暂的一生。

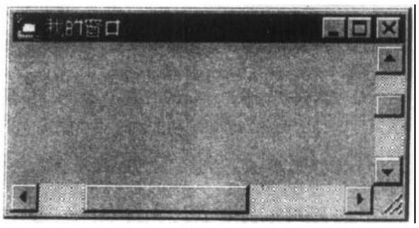


图 2-2　用 TWindow 创建的窗口对象

如果你想要窗口在消失之前做点什么，你就需要在窗口的界面元素被删除之前增加一些程序代码。

```
//构造一个 Twindow 对象:
TWindow * AWindow;
AWindow = new TWindow(0,"我的窗口");

//设置窗口的属性:
//窗口风格设置:
AWindow->Attr.Style |= WS_BORDER | WS_CAPTION | WS_CHILD | WS_HSCROLL | WS_MAXIMIZEBOX | WS_MINIMIZEBOX | WS_SYSMENU | WS_VISIBLE | WS_VSCROLL;

//窗口扩展风格设置:
AWindow->Attr.ExStyle |= WS_EX_TOPMOST;

//窗口的位置和大小:
AWindow->Attr.X =10;
AWindow->Attr.Y =20;
AWindow->Attr.W = 80;
AWindow->Attr.H = 20;

//设置窗口的背景颜色:
AWindow->SetBkgndColor(RGB(240,252,128));

//创建与窗口对象相关的窗口界面元素:
AWindow->Create();

//显示窗口
AWindow->Show(SW_SHOW);

//在这里加入你自己的程序代码
⋮
//删除窗口的界面元素:
AWindow->Destroy();

//删除这个窗口对象:
delete AWindow;
```

这段代码并不是完整的程序，但它们却完成了窗口对象从创建到删除的整过程。

另外，如果一个窗口是应用程序的主窗口，那么应用程序运行开始时将自动创建并显示它；应用程序结束时当然也会自动删除它。

2.2 从最简单的程序开始

也许你总是抱怨编写 Windows 程序是多么罗嗦，即使是最简单的 Windows 程序也需要写一大堆代码，你要定义一个回调函数、在 WinMain 函数中定义窗口、设置窗口的风格、创建窗口、显示窗口以及进行消息循环。是啊，真罗嗦！

现在，我们要用 OWL 5.0 来实现同样的功能(其实什么功能都没有)。让我们欢迎第一个 OWL 程序出场，它是最简单但同时也是一个完整的 Windows 应用程序。而且这个程序确实已经运行通过了。

```
#include <OWL\pch.h>
int OwlMain(int , char * [])
{
      return TApplication().Run();
}
```

在这个程序中，我们用 TApplication 类创建一个应用程序，这个应用程序具有一个缺省的 Windows 窗口，可以最大化、最小化和移动。你不需要为 ObjectWindows 应用程序显式地给出 WinMain 函数，而使用替代函数 OwlMain，OwlMain 可以使用在传统 C 和 C++程序的 main 函数中用到的参数 int argc 和 char * * argv。

当然，我们学习 OWL 并不是想要编写这样的简单程序，我们需要窗口能够完成一些事情(比如绘图或写字)，还需要我们的应用程序和其他流行的 Windows95 应用程序一样具有菜单条(比较复杂的那种)、工具条、对话框、包含子窗口、可以响应许多事件，最后，还要一个漂亮的图标。这些都是 Windows 95 应用程序的基本要求，而通过 OWL 都可以得到圆满的解决。

2.3 应用程序和 TApplication 类

ObjectWindows 用 TApplication 和 TModule 类封装了 Windows 的应用程序和 DLL 模块。TModule 类封装了 Windows DLL 的初始化和关闭功能，也包含了 hInstance 和 lpCmdLine 两个参数，这两个参数与非 OWL 应用程序的 WinMain 中同名的两个参数有着相同的意义。除非你要用 DLL，否则你是不需要自己建立 TModule 类的。以下，我们主要讨论 TApplication 类。

一个应用程序能够运行，它所需要的条件是：

1. 在包含文件中加入正确的 ObjectWindows 头文件。TApplication 的定义在头文件 OWL\applicat.h 中，使用 TApplication 必须包含这个头文件。当然，就正如你在前面的“最简单的程序”中看到的那样，如果用 OWL 5.0 创建应用程序，那么，CPP 文件中只要包含 OWL\pch.h 头文件就可以省去这些麻烦。

2. 建立一个 ObjectWindows 应用程序对象。可以通过两个构造函数来建立应用程序对象，通常用的构造函数是

```
TApplication(const char far * name = 0,
        TModule * & gModule = ::Module,
        TAppDictionary * appDict = 0);
```

Tapplication 的这个构造函数包含一个字符串，它将成为应用程序的名字。如果你不给出一个名字，构造函数将使用一个缺省的空字符串。gModule 参数可以让你设置应用程序的全局指针;appDict 参数设置应用程序所使用的辞典，如果不设置，它将使用缺省值。

TApplication 的另外一个构造函数中包括了 WinMain 函数中的参数。

```
TApplication(const char far * name,
        HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        const char far * cmdLine,
        int cmdShow,
        TModule * & gModule = ::Module,
        TAppDictionary * appDict = 0);
```

你可以使用构造函数传递参数来建立一个应用程序对象。

3. 调用 Run 函数。显然，TApplication::Run 的功能是运行一个应用程序，但是在此期间它还完成了一系列的任务，其中包括:初始化 ObjectWindows 应用程序，建立和显示 ObjectWindows 主窗口，运行应用程序的消息循环。当然，对于建立的基础的应用程序，你只需要了解 Run 函数是使你的程序运行的函数就够了。

TApplication 包括了三个初始化函数，它们是:InitApplication 函数初始化应用程序的第一个实例；InitInstance 函数初始化应用程序的每一个实例;InitMainWindow 函数初始化应用程序的主窗口。

这些函数是否被调用取决于它是否是应用程序的第一个实例。对于 16 位的应用程序，InitApplication 函数只被第一个实例调用;InitInstance 函数是第一个实例调用的下一个函数，但它却是被增加的实例调用的第一个函数;InitInstance 调用 InitMainWindow 函数。在 32 位应用程序中，每一个应用程序在它自己的内存地址中运行，它们彼此不共享实例代码，使得所有的实例都是第一个实例。所以你每次启动 32 位应用程序，都要运行所有的初始化函数。如果当前的实例是第一个实例(表现为数据成员 hPrevInstance 被设置为 0)，InitApplication 函数就被调用。你可以通过重载 InitApplication 函数来完成一些任务，因为缺省的 InitApplication 函数是不做任何事的。例如，你可以用 InitApplication 函数在主窗口上标志是否为第一个应用程序，只要在派生类里增加一个 WindowTitle 数据成员，将其设置为“增加的应用程序实例”，然后重载 InitApplication 函数设置 WindowTitle 为"应用程序的第一个实例”。如果应用程序是第一个实例，就会调用重载的 InitApplication 函数。

另外，应用程序对象包含了方便地调用 BWCC (Borland Custom Controls Library)和三维控件库(Microsoft 3 - D Controls Library)的功能，还有一个把标准控件自动转换为 Microsoft 3 - D 控件的函数。

2.4 具有主窗口的应用程序

前面我们编写了一个最简单的应用程序，但它什么都不做，甚至连窗口的标题都没有。缺省的 TApplication::InitMainWindow 函数会建立一个与应用程序对象名字相同的主窗口。这个窗口不是非常有用，因为它不接收和处理任何的用户输入。你必须重载 InitMainWindow 函数来建立一个窗口对象响应用户的输入。在重载的 InitMainWindow 函数中建立一个派生于 TWindow 类或 TFrameWindow 类的对象，并且调用应用程序类的一个成员函数 SetMainWindow。SetMainWindow 需要一个指向 TFrameWindow 类的指针参数并返回一个指向旧主窗口的指针(如果它是一个以前没有建立过主窗口的新的应用程序，返回值就为 0)。

下面的例子显示了一个简单的应用程序，其中建立 TFrameWindow 对象作为应用程序的主窗口：

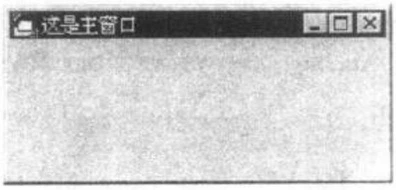


图 2-3 包含主窗口的应用程序

```
#include <owl\pch.h>
class TMyApp : public TApplication {
  public:
      TMyApp() : TApplication() {}

    void InitMainWindow()
    {
        SetMainWindow(new TFrameWindow(0, "这是主窗口"));
    }
};
int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TMyApp().Run();}
```

当你运行这个应用程序时，主窗口标题就显示“这是主窗口”(见图 2-3)。这个例子什么也不做，它还是不提供处理用户输入的功能。

2.5 加入客户窗口

TFrameWindow 类提供框架窗口对象，框架窗口中可以包含一个客户窗口。框架窗口和客户窗口是 MDI(多文档界面)和 SDI(单文档界面)窗口的基础，在 ObjectWindows 环境中框架窗口具有特殊的地位：它们管理着应用程序的诸如菜单和工具条之类的功能。框架窗口中的客户窗口可用来完成特定的任务，你对框架窗口所做的改动(比如增加工具条和状态条等)不会对客户窗口产生影响。

TFrameWindow 是从 TWindow 类派生出来的，它控制着窗口的特定行为(比如键盘控制和客户窗口的命令执行)。当一个窗口被重新激活时，TFrameWindow 作出响应，恢复窗口的输入焦点并增加菜单条和图标。TFrameWindow 类是可流类。在窗口的一系列区域中，框架窗口的区域包括边界、系统菜单、工具条和状态条、其它就属于客户窗口区域。虽然

框架窗口可以支持客户窗口，但它与客户窗口保持相对独立，你可以改变客户窗口而不会影响框架窗口。

下面的程序创建了一个具有客户窗口的主窗口。

```
#include <owl\pch.h>
class TMyClientWindow : public TWindow
{
public:
  TMyClientWindow(TWindow * parent = 0, const char far * title = 0, TModule * module = 0);
};

TMyClientWindow :: TMyClientWindow(TWindow * parent, const char far * title, TModule * module)
      : TWindow(parent, title, module)
{
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;
}

class TMyApp : public TApplication
{
public:
  TMyApp() : TApplication() {}
  void InitMainWindow();
  };

void TMyApp :: InitMainWindow()
{
     SetMainWindow(new TFrameWindow(0, "加入客户窗口", new TMyClientWindow));
}

int
OwlMain(int /* argc */, char * /* argv */ [])
{
  return TMyApp().Run();
}
```

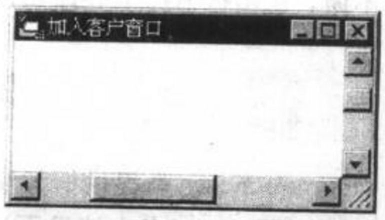


图 2-4 加入客户窗口的主窗口

正如你看到的(见图 2-4)，这个窗口具有两个滚动条，而它们是由客户窗口提供的。

在 ObjectWindows 体系中用到这种 frame/client (框架窗口/客户窗口)结构的还有 TMDIChild 类，它也可以包括一个客户窗口。为客户区域提供单独的类可以使你的程序变得更加灵活，通常框架窗口响应一些关于窗口的操作(打开或关闭窗口、设置菜单条等)，而客户窗口则将注意力集中到程序的功能上(比如在窗口中绘图)。你还可以把一个对话框作为客户窗口放到其它的框架窗口中。

忠告：当你看到一个 Windows 窗口时，就应该象“庖丁解牛”一样把它看作是框架窗口

/客户窗口结构的两个窗口。

2.6 让窗口响应事件

应用程序初始化以后，应用程序的消息循环就开始工作了，消息循环响应和分配来自 Windows 系统的消息。对于前面列出的那个具有客户窗口的程序，我们需要客户窗口响应一些简单的消息，包括鼠标的左键按下和右键按下，我们就要在客户窗口类中定义下面的响应事件表：

```
class TMyClientWindow : public TWindow
{
...
  DECLARE_RESPONSE_TABLE(TMyClientWindow);
};

DEFINE_RESPONSE_TABLE1(TMyClientWindow, TWindow)
  EV_WM_LBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
END_RESPONSE_TABLE;
```

在这个响应事件表中，我们列出了两个事件，它们就是这个客户窗口需要响应的所有事件。当然我们也可以用同样的方法来定义框架窗口类和应用程序类的响应事件表。对于不同的事件，窗口的响应当然是不同的，这就需要定义不同的响应函数。对于像"鼠标左键按下事件"(EV_WM_LBUTTONDOWN)和"鼠标右键按下事件"(EV_WM_RBUTTONDOWN)这样的标准事件，TWindow 类已经规定了它们的响应函数名称：

```
void EvLButtonDown(uint, TPoint&);
void EvRButtonDown(uint, TPoint&);
```

在派生的窗口中通过定义同名的函数来取代父类的响应方式。

下面是实现窗口的事件响应的程序：

```
#include <owl\pch.h>
class TMyClientWindow : public TWindow
{
public:
  TMyClientWindow(TWindow * parent = 0, const char far * title = 0, TModule * module = 0);
      void EvLButtonDown(uint, TPoint&);
      void EvRButtonDown(uint, TPoint&);
  DECLARE_RESPONSE_TABLE(TMyClientWindow);
};

DEFINE_RESPONSE_TABLE1(TMyClientWindow, TWindow)
  EV_WM_LBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
```

```
END_RESPONSE_TABLE;

void TMyClientWindow::EvLButtonDown(uint, TPoint&)
{
  MessageBox("鼠标左键按下", "信息", MB_OK);
}

void TMyClientWindow::EvRButtonDown(uint, TPoint&)
{
  MessageBox("鼠标右键按下", "信息", MB_OK);
}

TMyClientWindow :: TMyClientWindow(TWindow * parent, const char far * title,
    TModule * module)
        : TWindow(parent, title, module)
{
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;
}

class TMyApp : public TApplication {
  public:
      TMyApp() : TApplication() {}
      void InitMainWindow();
};

void TMyApp :: InitMainWindow()
{
      SetMainWindow(new TFrameWindow(0, "加入事件响应", new TMyClientWindow));
}

int OwlMain(int /* argc */, char * /* argv */ [])
{
  return TMyApp().Run();
}
```

图 2-5　客户窗口响应鼠标事件

上面的应用程序中客户窗口对于鼠标事件的响应非常简单，它只是弹出了一个标准的消息窗口(见图 2-5)，对应的语句为：

MessageBox("鼠标左键按下", "信息", MB_OK);

我们当然可以根据实际应用的情况对事件作出特殊的处理。不要忘记，Windows 程序本身就是以事件驱动的方式运行的。

2.7 资源与资源编辑器

资源是应用程序中的一些只读数据，它包括加速键、位图、光标、对话框、字体、图标、菜单、字符串和用户自定义资源。它们的意义如表 2-2 所示。

表 2-2

资 源	意 义
加速键	应用程序中的热键
位图	图形映象的二进制形式
光标	指示鼠标的位置
对话框	包含各种控件，允许用户选项
字体	定义字体的字样、大小和格式
图标	应用程序最小化的位图
菜单	提供选择，可以发出命令
字符串	定义应用程序需要的字符串

每一个资源数据(比如一个对话框，一个图表或是一个菜单)都有一个资源 ID 值与其对应，ID 值具有整数类型。资源 ID 值就是这项资源的唯一标识(也就是我们给某个资源项取的名字)，在同一种资源中不能有两个相同的 ID 值(不能有两个图标的 ID 值都是 100)；但在不同的资源类型中可以有相同的 ID 值，比如一个图标的 ID 值是 100，而一个对话框的 ID 值也可以是 100。有时需要在不同资源类型中定义相同的 ID 值，并利用它们巧妙地完成一些“特殊任务”，这一技巧将在下面的章节中介绍。为了便于使用资源 ID 值，程序中往往定义一个宏来帮助记忆，比如：

```
#define MENU_1  100
```

这种做法非常普遍，它可以避免直接使用整数值带来的失误或麻烦。

大多数应用程序都需要使用一些资源，我们可以用集成在 BC++ 5.0 中的资源编辑器(Resource WorkShop)来创建这些资源(见图 2-6)，然后将它们存放在资源文件中。资源编辑器将资源文件与编译好的可执行文件相连接就产生了一个 Windows 应用程序。

下面我们要在资源文件中加入菜单、图标和加速键表。我们还需要做一些准备工作，定义一些整数宏作为资源 ID 值。所有的定义都放到 prj.rh 头文件中，然后在资源文件和 CPP 文件中都包含这个头文件。

```
//prj.rh 头文件
#define ACCELERATORS_1  1
#define MENU_1  1
#define ICON_1  1

#define CM_OPEN  101
#define CM_CLOSE  102
```

```
#define CM_ABOUT  103
#define CM_EXIT  24310
```

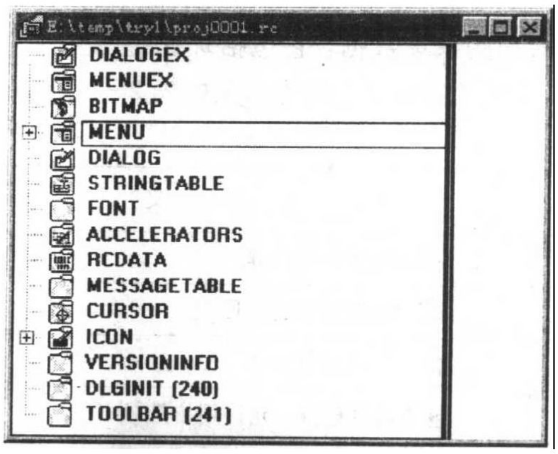


图 2-6　集成环境中的资源编辑器

首先，我们在资源编辑器中加入菜单，这是通过菜单编辑器进行的(见图 2-7)。具体步骤如下：

1. 双击资源编辑器中的 menu 项，弹出 New menu resource options 对话框；

2. 在对话框中的 Type of menu resource to create 项目中选择 Simple menubar 以创建一个新的菜单；在 Name 项目中可以给出菜单的资源 ID 值。如果确认就点击 OK 按钮，打开菜单编辑器；

3. 菜单编辑器中已经给出了一个弹出菜单(Popup)和一个菜单项(Menuitem)，你需要做的就是修改它们并且增加新的弹出菜单和菜单项。Text 项目中可以输入需要显示的文字，文字中加入"&"符号是为了显示菜单的加速键(比如"&File")，对于中文菜单，你不要试图在汉字前面加"&"符号，你可以采用"文件(&F)"的方法。如果希望灵活地使用菜单编辑器，你就需要经常的点击鼠标右键或是拖动鼠标，这可能会给你带来新的发现。

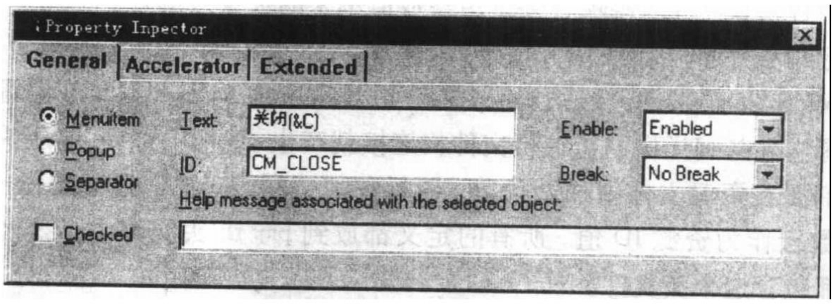


图 2-7　在资源编辑器中编辑菜单

下面是菜单编辑器所生成的代码：

```
MENU_1 MENU
{
  POPUP "文件(&F)"
  {
```

```
    MENUITEM "打开(&O)", CM_OPEN
    MENUITEM "关闭(&C)", CM_CLOSE
    MENUITEM SEPARATOR
    MENUITEM "退出(&X)", CM_EXIT
  }

  POPUP "帮助(&H)"
  {
    MENUITEM "关于(&A)", CM_ABOUT
  }

}
```

在资源编辑器中编辑图标是通过绘图编辑器实现的(见图 2-8)。

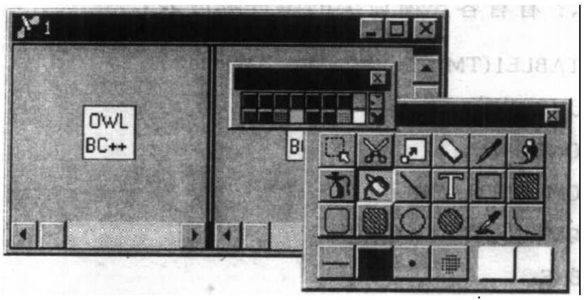


图 2-8　在资源编辑器中绘制图标

用加速键表编辑器编辑加速键表，得到加速键表代码，下面代码定义的加速键都是 ASCII 码与 Ctrl 键的组合：

```
ACCELERATORS_1 ACCELERATORS
{
  "^o", CM_OPEN, ASCII
  "^c", CM_CLOSE, ASCII
  "^e", CM_EXIT, ASCII
  "^a", CM_ABOUT, ASCII
}
```

2.8　事件响应程序实例

通过前面的程序我们对窗口如何进行消息处理已经有初步的了解，知道了如何定义窗口要响应一些标准的消息。这一节我们主要通过程序实例学习如何使用一些最基本的资源，包括：菜单、图标和加速键表。为了更好的理解应用程序事件响应的方法，我们在程序中定义了应用程序类的事件响应表、框架窗口类的事件响应表和已经见过的客户窗口的事件响应表。

我们先看看客户窗口类的定义：

```
class TMyClientWindow : public TWindow
{
public:
  TMyClientWindow(TWindow * parent = 0, const char far * title = 0, TModule * module = 0);
  void EvLButtonDown(uint, TPoint&);
  void EvRButtonDown(uint, TPoint&);
  void CmOpen();
  void CmClose();
    DECLARE_RESPONSE_TABLE(TMyClientWindow);
};
```

与前面的客户窗口类相比，现在增加了两个成员函数：

void CmOpen();

void CmClose();

它们是功能是什么？看看客户窗口类的事件响应表：

```
DEFINE_RESPONSE_TABLE1(TMyClientWindow, TWindow)
    EV_WM_LBUTTONDOWN,
    EV_WM_RBUTTONDOWN,
    EV_COMMAND(CM_OPEN, CmOpen),
    EV_COMMAND(CM_CLOSE, CmClose),
END_RESPONSE_TABLE;
```

事件响应表中定义了两个菜单项命令的响应函数，EV_COMMAND 宏的用法是：

EV_COMMAND(菜单项命令，响应函数)

这样，新增的两个函数就可以响应两个菜单项命令了。

在前面的程序中我们直接使用 TFrameWindow 类来定义框架窗口，为了让框架窗口能够响应事件，我们派生出一个新的框架窗口类。我们看看它的定义：

```
class TMainWindow : public TFrameWindow
{
public:
  TMainWindow(TWindow * parent = 0, const char far * title = 0, TWindow * clientWnd = 0,
bool shrinkToClient = false, TModule * module = 0);
  virtual void SetupWindow ();
  void CmQuit();
  DECLARE_RESPONSE_TABLE(TMainWindow);
};
```

框架窗口类的事件响应表：

```
DEFINE_RESPONSE_TABLE1(TMainWindow, TFrameWindow)
    EV_COMMAND(CM_QUIT, CmQuit),
END_RESPONSE_TABLE;
```

不难看出，框架窗口类所响应的事件是菜单项命令 CM_QUIT，而 CmQuit 函数就是这

个菜单项命令的响应函数。

由于与框架窗口类同样的原因，我们重新定义了一个应用程序类：

```
class TMyApp : public TApplication {
  public:
      TMyApp() : TApplication() {}
      void InitMainWindow();
      void CmAbout();
  DECLARE_RESPONSE_TABLE(TMyApp);
};

DEFINE_RESPONSE_TABLE1(TMyApp, TApplication)
  EV_COMMAND(CM_ABOUT, CmAbout),
END_RESPONSE_TABLE;
```

从事件响应表中可以看出：CmAbout 函数响应菜单项命令 CM_ABOUT。

对于菜单和图标资源，TFrameWindow 类提供了一系列成员函数可以直接进行操作，见表 2-3。

表 2-3

成员函数	意义解释
AssignMenu	这是在界面元素创建之前设置窗口菜单的典型函数，你可以在 InitMainWindow 函数或窗口对象的构造函数以及 SetupWindow 函数中调用
SetMenu	将窗口的菜单句柄设置到 HMENU 参数中
SetMenuDescr	将窗口的菜单描述设置到 TMenuDescr 参数中
GetMenuDescr	返回当前的菜单描述
MergeMenu	将当前菜单与 TMenuDescr 参数所描述的菜单融合
RestoreMenu	从 Attr. Menu 中恢复窗口菜单
SetIcon	以第一个参数的方式设置图标，第二个参数确定资源

表 2-3 中的 bool SetIcon(TModule * iconModule, TResId iconResId);
可以设置窗口的图标，但它需要一个指向模块对象(或应用程序对象)的指针。如何在窗口中获得当前应用程序对象的指针呢?

有时，你需要从应用程序类外访问应用程序(比如从派生的窗口类的成员函数里调用应用程序对象的成员)，但是因为窗口对象不在应用程序对象的范围里，所以通常不能直接访问应用程序对象的成员。若要访问就需要得到应用程序对象指针。TWindow 类提供了一个成员函数 GetApplication，它返回一个指向应用程序对象的指针。利用这个指针你就可以访问应用程序的成员函数和数据成员了。程序中就使用了一个窗口类的成员函数：

```
void TMainWindow :: SetupWindow ()
{
......
```

```
        //
        // Assign ICON w/ this application.
        //
        SetIcon(GetApplication(), ICON_1);
  ......
  }
```

你可以用三种方法来装载资源文件中的加速键表：在主窗口中说明；在主窗口的 SetupWindow 函数中说明；在应用程序类的 InitMainWindow 函数中说明。具体可见下面的示例。

```
 * 在主窗口的构造函数中说明
TMainWindow :: TMainWindow(TWindow * parent , const char far * title,
        TWindow * clientWnd, bool shrinkToClient, TModule * module)
        :TFrameWindow(parent, title, clientWnd , shrinkToClient, module)
{
......
        Attr.AccelTable = ACCELERATORS_1;
......
}
 * 在主窗口的 SetupWindow 函数中说明
void TMainWindow :: SetupWindow ()
{
......
        Attr.AccelTable = ACCELERATORS_1;
        LoadAcceleratorTable();
......
}
 * 在应用程序类的 InitMainWindow 函数中说明：
void TMyApp :: InitMainWindow()
{
......
    TFrameWindow *frame = new TMainWindow(0, "加入资源", new TMyClientWindow);
    frame-> Attr.AccelTable = ACCELERATORS_1;
    SetMainWindow(frame);
......
}
```

编写 Windows 程序时最好建立一个项目文件 *.ide(见图 2-9)，在项目文件中可以包括一系列 CPP 源程序、资源文件和定义文件。当然，在一个项目文件中也可以包括若干个目标文件 *.exe(或 *.dll)。

程序编译执行后，你可以检查一下它对鼠标、菜单项以及热键的响应情况(见图 2-10)。另外，你可以试着把响应表中对 CM_EXIT 的定义去掉，看看执行的结果。当点击菜单项“退出”时，程序便退出了。是计算机的智能吗？当然不是！是资源 ID 值 CM_EXIT 的值 24310 起作用了，因为 OWL 类库中已经定义了它的响应函数的作用就是退出程序。

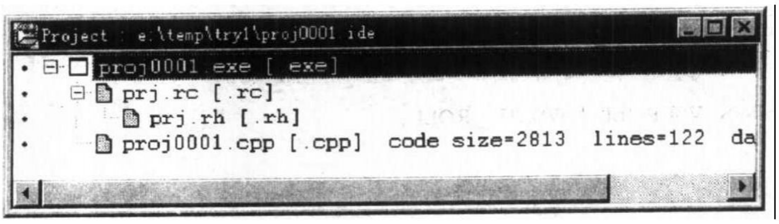


图 2-9　应用程序的项目文件

下面是完整的一个应用程序代码：

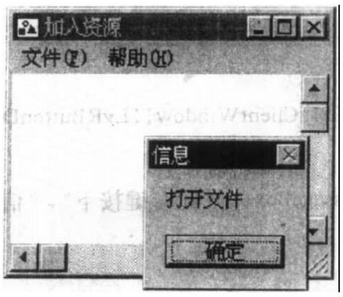


图 2-10　在应用程序中使用资源

```
//———————————————————
//              resource.cpp
//———————————————————
#include <owl\pch.h>

#include "PRJ.rh"

class TMyClientWindow : public TWindow
{
  public:
      TMyClientWindow(TWindow * parent = 0, const char far * title = 0, TModule * module = 0);
      void EvLButtonDown(uint, TPoint&);
      void EvRButtonDown(uint, TPoint&);
      void CmOpen();
      void CmClose();   DECLARE_RESPONSE_TABLE(TMyClientWindow);
};

class TMainWindow : public TFrameWindow
{
  public:
      TMainWindow(TWindow * parent = 0, const char far * title = 0, TWindow * clientWnd = 0,
bool shrinkToClient = false, TModule * module = 0);
      virtual void SetupWindow ();
      void CmExit();
  DECLARE_RESPONSE_TABLE(TMainWindow);
};

DEFINE_RESPONSE_TABLE1(TMyClientWindow, TWindow)
  EV_WM_LBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
  EV_COMMAND(CM_OPEN, CmOpen),
  EV_COMMAND(CM_CLOSE, CmClose),
END_RESPONSE_TABLE;
```

```
TMyClientWindow :: TMyClientWindow(TWindow * parent, const char far * title, TModule * module)
        : TWindow(parent, title, module)
{
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;
}

void TMyClientWindow::EvLButtonDown(uint, TPoint&)
{
  MessageBox("鼠标左键按下", "信息", MB_OK);
}

void TMyClientWindow::EvRButtonDown(uint, TPoint&)
{
  MessageBox("鼠标右键按下", "信息", MB_OK);
}

void TMyClientWindow::CmOpen()
{
  MessageBox("打开文件", "信息", MB_OK);
}

void TMyClientWindow::CmClose()
{
  MessageBox("关闭文件", "信息", MB_OK);
}

DEFINE_RESPONSE_TABLE1(TMainWindow, TFrameWindow)
  EV_COMMAND(CM_EXIT, CmExit),
END_RESPONSE_TABLE;

TMainWindow :: TMainWindow(TWindow * parent , const char far * title, TWindow * clientWnd, bool
shrinkToClient, TModule * module)
        :TFrameWindow(parent, title, clientWnd , shrinkToClient, module)
{
}

void TMainWindow :: SetupWindow ()
{
  TFrameWindow::SetupWindow ();
      //
      // Assign ICON w/ this application.
      //
      SetIcon(GetApplication(), ICON_1);
```

```
    //
    // Menu associated with window and accelerator table associated with table.
    //
    AssignMenu(MENU_1);
    //
    // Associate with the accelerator table.
    //
    Attr.AccelTable = ACCELERATORS_1;
    LoadAcceleratorTable();
}

void TMainWindow::CmExit()
{
  MessageBox("退出程序", "信息", MB_OK);
}

//----------------------------------------------------------
class TMyApp : public TApplication {
  public:
    TMyApp() : TApplication() {}
    void InitMainWindow();
    void CmAbout();
  DECLARE_RESPONSE_TABLE(TMyApp);
};

DEFINE_RESPONSE_TABLE1(TMyApp, TApplication)
  EV_COMMAND(CM_ABOUT, CmAbout),
END_RESPONSE_TABLE;

void TMyApp :: InitMainWindow()
{
    TFrameWindow *frame = new TMainWindow(0, "加入资源", new TMyClientWindow);

    SetMainWindow(frame);
}

void TMyApp::CmAbout()
{
}

int OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TMyApp().Run();
```

2.9 AppExpert 和 ClassExpert

使用 OWL 5.0 编写应用程序确实方便了许多，但你也许还觉得自己书写全部代码有些麻烦，而且容易出错。基于 OWL 5.0 的编程工具 AppExpert 和 ClassExpert 可以帮助你建立 OWL 应用程序的框架并且对程序进行有效地扩充，在这方面，它们确实是专家。

AppExpert 是 BC++ 5 中专门用于创建应用程序框架的工具，创建新的应用程序时，选择 file/new/appexpert 菜单项，即可运行 AppExpert 。

在主项 Application 中可以定义应用程序的基本结构：MDI(多文档界面)、SDI(单文档)和 Dialog Client(对话框框架)，还可选择是否采取 Document/View(文档/视图)结构。在子项 Basic Options 中可以选择文件路径、目标文件名、帮助文件名，还可选择可移动工具条、状态条、快捷文件列、支持注册、文件拖放、打印和电子邮件等功能；在 Advanced Options 子项中可以选择主窗口打开时的状态(包括普通、最小化和最大化状态)，选择控件的风格(标准、BWCC 风格、MS 3D 风格)；在 OLE 2 Options 子项中选择应用程序对 OLE 客户方和服务方的支持，也可选择对 OCX 的支持；Code Gen Control 子项中设置自动生成的程序状态；在 Admin Options 子项中进行版本控制。

在主项 Main Window 中可以选择主窗口的标题和背景颜色。在子项 Basic Options 中可以设置主窗口的风格：标题、边框、最大化按钮、最小化按钮、纵横滚动条等；如果程序结构是单文档，则在 SDI Client 子项中选择单文档客户窗口的风格；如果程序结构是多文档，则在 MDI Client 子项中选择多文档客户窗口的风格；如果程序是对话框结构，则在 Dialog Client 子项中选择对话框客户窗口的风格。

在主项 MDI Child/View 和它的子项 Basic Options 中选择多文档结构中子窗口的框架和视图的属性。

2.10 AppExpert 程序实例

为了说明 AppExpert 的具体使用方法并为下一章介绍 Windows 控件提供方便，我们使用 AppExpert 来生成一个以对话框为基础的应用程序框架，取名为 OWL。首先在 Application 主项中选择 Dialog Client，去掉 Document/View 选项；然后在子项中去掉所有的无关选项(或设置为缺省值)；在 Main Window 主项 Dialog Client 子项中设置对话框的类名、文件名等，也可使用缺省值；最后点击 Generate 按钮便可生成所要的应用程序框架(见图 2-11)。实际上所生成的代码是一个完整的程序，我们可以将它编译、执行(见图 2-12)。

下面是自动生成的代码：

```
//-----------------------------------------------------------------
// Project Owl
// FILE:          OwlApp.h
// Class definition for TOwlApp (TApplication).
//-----------------------------------------------------------------
```

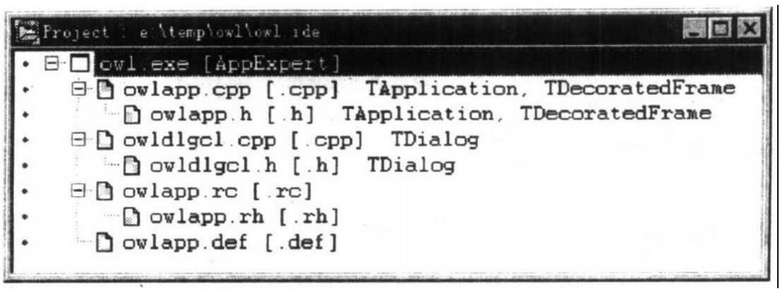


图 2-11　用 AppExpert 创建应用程序框架

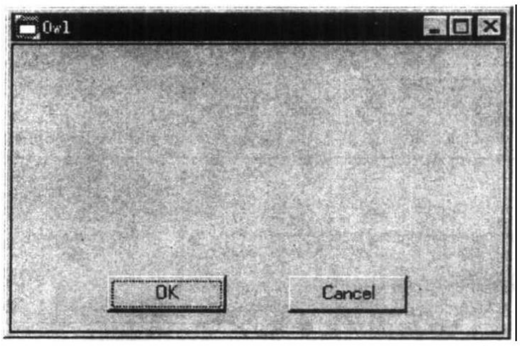


图 2-12　应用程序执行情况

```
#if ! defined(owlapp_h)              // Sentry, use file only if it's not already included.
#define owlapp_h
#include <owl/opensave.h>
#include "OwlApp.rh"              // Definition of all resources.

//
// FrameWindow must be derived to override Paint for Preview and Print.
//
//{{TDecoratedFrame = TSDIDecFrame}}
class TSDIDecFrame : public TDecoratedFrame {
  public:
    TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool trackMenuSe-
lection = false, TModule * module = 0);
    ~TSDIDecFrame();
};        //{{TSDIDecFrame}}

//{{TApplication = TOwlApp}}
class TOwlApp : public TApplication {
  private:


  public:
    TOwlApp();
    virtual ~TOwlApp();
```

```
//{{TOwlAppVIRTUAL_BEGIN}}
  public:
    virtual void InitMainWindow();
//{{TOwlAppVIRTUAL_END}}

//{{TOwlAppRSP_TBL_BEGIN}}
  protected:
    void CmHelpAbout();
//{{TOwlAppRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TOwlApp);
};        //{{TOwlApp}}
#endif // owlapp_h sentry.

//------------------------------------------------------------
// Project Owl
// FILE:          OwlDlgCl.h
// Class definition for TOwlDlgClient (TDialog).
//------------------------------------------------------------
#if !defined(owldlgcl_h)            // Sentry, use file only if it's not already included.
#define owldlgcl_h
#include "OwlApp.rh"              // Definition of all resources.

//{{TDialog = TOwlDlgClient}}
class TOwlDlgClient : public TDialog {
  public:
    TOwlDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TOwlDlgClient();

};        //{{TOwlDlgClient}}

#endif     // owldlgcl_h sentry.

//------------------------------------------------------------
// Project Owl
// FILE:          OwlApp.rc
// Constant definitions for all resources defined in OwlApp.rc.
//------------------------------------------------------------
//#if !defined(owlapp_rh)      // Sentry use file only if it's not already included.
//#define owlapp_rh

......
```

```
//#endif      // owlapp_rh

//-----------------------------------------
// Project Owl
// FILE:          OwlApp.cpp
// Source file for implementation of TOwlApp (TApplication).
//-----------------------------------------
#include <owl/pch.h>
#include <stdio.h>
#include "OwlApp.h" #include "OwlDlgCl.h"          // Definition of client class.

//{{TOwlApp Implementation}}
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TOwlApp, TApplication)
//{{TOwlAppRSP_TBL_BEGIN}}
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
//{{TOwlAppRSP_TBL_END}}
END_RESPONSE_TABLE;

//-----------------------------------------
// TOwlApp
//
TOwlApp::TOwlApp() : TApplication("Owl")
{

  // INSERT>> Your constructor code here.
}

TOwlApp::~TOwlApp()
{
  // INSERT>> Your destructor code here.
}

//-----------------------------------------
// TOwlApp
// ~~~~~
// Application intialization.
//
void TOwlApp::InitMainWindow()
{
```

```
    if (nCmdShow != SW_HIDE)
      nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

    TSDIDecFrame* frame = new TSDIDecFrame(0, GetName(), 0, false);
    frame->SetFlag(wfShrinkToClient);

    // Assign ICON w/ this application.
    //
    frame->SetIcon(this, IDI_SDIAPPLICATION);
    frame->SetIconSm(this, IDI_SDIAPPLICATION);

    SetMainWindow(frame);

  }

  //{{TSDIDecFrame Implementation}}

  TSDIDecFrame::TSDIDecFrame(TWindow* parent, const char far* title, TWindow* clientWnd, bool
trackMenuSelection, TModule* module)
  :
    TDecoratedFrame(parent, title, !clientWnd ? new TOwlDlgClient(0) : clientWnd, trackMenuSelection,
module)
  {  // INSERT>> Your constructor code here.
  }

  TSDIDecFrame::~TSDIDecFrame()
  {
    // INSERT>> Your destructor code here.
  }

  //-------------------------------------------------
  // TOwlApp
  // ~~~~~~~~~~~
  // Menu Help About Owl command
  //
  void TOwlApp::CmHelpAbout()
  {
  }

  int OwlMain(int , char* [])
  {
    TOwlApp app;
    return app.Run();
```

```
}

//------------------------------------------------------------------------
// Project Owl
// FILE:          OwlDlgCl.cpp
// Source file for implementation of TOwlDlgClient (TDialog).
//------------------------------------------------------------------------
#include <owl/pch.h>
#include "OwlApp.h"
#include "OwlDlgCl.h"

//{{TOwlDlgClient Implementation}}

//------------------------------------------------------------------------
// TOwlDlgClient
// ~~~~~~~~~~~
// Construction/Destruction handling.
//
TOwlDlgClient::TOwlDlgClient(TWindow* parent, TResId resId, TModule* module)
:
  TDialog(parent, resId, module)
{
  // INSERT>> Your constructor code here.
}

TOwlDlgClient::~TOwlDlgClient()
{
  Destroy();

  // INSERT>> Your destructor code here.
}

//------------------------------------------------------------------------
// Project Owl
// FILE:          OwlApp.rc
// All resources defined here.
//------------------------------------------------------------------------
#if !defined(WORKSHOP_INVOKED)
# include <windows.h>
#endif
#include "OwlApp.rh"
......
```

```
// Application ICON
//
IDI_SDIAPPLICATION ICON "applsdi.ico"

// Client Dialog
//
IDD_CLIENT DIALOG 6, 15, 207, 111
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Dialog Client"
FONT 8, "MS Sans Serif"
BEGIN
  DEFPUSHBUTTON "OK", IDOK, 40, 88, 50, 14
  PUSHBUTTON "Cancel", IDCANCEL, 116, 88, 50, 14
END
```

ClassExpert 是另外一个 OWL 应用程序工具，它可以帮助程序员维护用 AppExpert 创建的程序代码(见图 2-13)。在 ClassExpert 中可以增加、修改、删除对象，增加窗口或控件响应的事件处理函数，当然也可以直接修改代码。使用 ClassExpert 的最重要的技巧在于使用鼠标右键，你可以在各种项目上点击鼠标右键，得到最快捷的操作。

本书中的大多数应用程序都将使用 AppExpert 和 ClassExpert，关于它们的更详细的用法和技巧请参照相应的联机帮助。

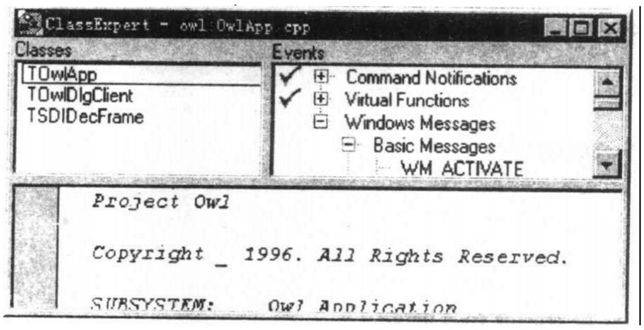


图 2-13　ClassExpert 的执行情况

第3章

控 件 类

3.1 什么是控件对象

控件对象是具有特殊行为的标准用户界面元素。除了 Windows 提供的标准控件以外，ObjectWindows 还提供了一系列用户控件。ObjectWindows 提供了这些控件的界面对象，你可以直接在程序中使用。控件的界面对象叫做控件对象。

表 3-1 列出了 ObjectWindows 提供的部分控件类。

表 3-1

控件		控件类名	描 述
标准的 Windows 控件	Push Button	TButton	标准按钮，带有文字
	Radio button	TRadioButton	Radio 按钮，带有文字可以设置开关量，常用于一个组中
	Check box	TCheckBox	检查框，带有文字可以设置开关量
	Static control	TStatic	静态文字或图片，用户不能改变
	Text Edit	TEdit	编辑框，用户可以在其中进行文字编辑
	List box	TListBox	列表框，可从中选择项目
	Combo box	TComboBox	组合框，带有一个列表框、编辑框或静态文字
	Scroll bar	TScrollBar	滚动条，带有一个活动方块和两个方向按钮
	Group box	TGroupBox	组，一个静态区域，左上角带有文字
Windows 通用控件	Animate	TAnimateCtrl	AVI 播放控件，可以播放 AVI 资源或 AVI 文件
	TabControl	TTabControl	对话框页面控件，在对话框中加入多页控件，可以切换
	Treeview	TTreeWindow	树型列表框，可显示类似目录树的数据
	Listview	TListWindow	列表显示框，可显示一系列图标和说明
	HotKey	THotKey	热键控件，可以定义与功能键结合的热键
	Gauge	TGauge	过程控件，显示一个过程的完成情况
	UpDown	TUpDown	增减控件，可以逐步改变一个值的大小
	Slider	TSlider	水平或垂直的范围选择控件，与滚动条类似
	RichEdit	TRichEdit	格式编辑框，可编辑带格式文本
	Header	TColumnHeader	标题头控件，可以说明表中列的内容，控制列的宽度
	StatusWindow	TStatusBar	状态条控件，在窗口底部显示多个状态值

在Windows中，控件是特殊的窗口；在ObjectWindows中，TControl类派生于TWindow类。控件对象和窗口对象作为子窗口时比较相像，它们也有相似的创建和删除过程。但是，标准的控件有别于其它的窗口，因为Windows处理窗口的事件消息并响应消息重画窗口；ObjectWindows用户控件却自己处理消息(ObjectWindows用户控件包含了重画和处理事件的代码)。

一个控件对象负责在父窗口中创建一个控件界面元素。在对话框中，控件对象可以将你的应用程序和对话框中的控件联系起来。

控件类具有两个构造函数，它们分别在窗口中和对话框中构造控件对象：

```
TControl(TWindow * parent,int id, const char far * title, int x, int y, int w, int h, TModule * module = 0);
TControl(TWindow * parent, int resourceId, TModule * module = 0);
```

一般来说，你可以直接使用控件类的实例(直接用控件类创建控件对象)，但是，有时你可能需要控件对象具有特殊的行为，于是创建了派生类。比如，从TListBox类中派生出TFontListBox类，其中包含了应用程序所能用到的所有字体。

不管使用的是那个控件对象，下面的几个步骤是必须的：

1. 创建控件；
2. 显示控件；
3. 删除控件。

创建控件

创建一个控件对象与创建一个其它类型子窗口一样，在父窗口的构造函数中调用子窗口的构造函数。除了通常的父窗口与子窗口之间的联系方式以外，控件与父窗口还有另外一种联系方式，就是调用通报。创建和初始化一个控件对象需要下列步骤：

1. 在父窗口中增加控件对象指针作为数据成员；
2. 调用控件对象的构造函数；
3. 改变控件属性；
4. 在SetupWindow中初始化控件。

显示控件

没有必要调用Windows函数Show来显示控件，控件是子窗口，它随着父窗口自动被显示和重画。当然你也可以调用Show来隐藏或显示控件。

删除控件

删除控件是父窗口的责任，当用户关闭窗口或应用程序时，控件的界面元素就随着父窗口自动删除。父窗口的析构函数自动删除它的全部子窗口。

建立与控件对象的联系

建立窗口对象与控件对象之间的联系同建立对话框对象与控件对象之间的联系很相似，窗口需要一种操作控件和响应控件事件的机制，比如响应列表框的选择消息。

设置和读取控件的值

想要管理一个包含多个子窗口的复杂对话框或窗口，你需要创建一个派生类来保存和恢复对话框或窗口的控件。控件的状态包括：编辑框的文字、滚动条的位置、Radio按钮的开关等等。另外一个办法是，你使用一个结构来描述对话框或窗口的控件状态，这个结构

叫做传输缓冲区(transfer buffer)。

每一种类型的控件操作起来都与其它类型的控件不同，下面将讨论怎样使用 Windows 标准控件对象和 ObjectWindows 用户控件对象。

3.2 按 钮 控 件

TButton 是代表按钮界面元素的界面类(见图 3-1)。在 TWindow 对象窗口中你必须使用 TButton 类在父窗口中创建一个按钮控件，你也可以在对话框中采用 TButton 对象用来建立应用程序与按钮之间的联系。

我们可以在资源编辑器中编辑一个对话框中的按钮，定义它的风格和 ID 值。

按钮被点击时完成一项任务，这就是按钮的功能。按钮有两种：缺省按钮和非缺省按钮。非缺省按钮具有细的边框，而缺省按钮具有粗的边框，它代表窗口的缺省动作。缺省按钮的风格参数为 BS_DEFPUSHBUTTON，非缺省按钮的风格参数为 BS_PUSHBUTTON。

用于窗口中的按钮控件构造函数是：

```
TButton(TWindow * parent,
        int resourceId,
        const char far * text,
        int X, int Y, int W, int H,
        bool isDefault = false,
        TModule * module = 0);
```

图 3-1 各种按钮控件

它除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还增加了按钮上的字符串和一个布尔量来标志是否为缺省按钮：

用于对话框中的按钮控件构造函数是：

```
TButton(TWindow * parent,
        int resourceID,
        TModule * module = 0);
```

它包含父窗口指针、资源 ID 值以及一个模块指针，其它的参数由资源文件决定。

下面是两个典型的按钮控件对象构造语句：

* 用于窗口的典型的按钮构造语句：

btn = new TButton(this, ID_BUTTON, "按钮", 38, 48, 316, 24, true);

* 用于对话框的按钮构造语句：

btn = new TButton(this, ID_BUTTON);

Borland 提供了 BWCC 风格的按钮，其中有几种预定义的按钮(见图 3-2)，它们的 ID 值和意义见表 3-2。

表 3-2

ID名	ID值	意义
IDOK	1	确定按钮
IDCANCEL	2	取消按钮
IDABORT	3	退出按钮
IDRETRY	4	重试按钮
IDYES	6	确认按钮
IDNO	7	否认按钮
IDHELP	998	帮助按钮

图 3-2　几种预定义的按钮

除了以上这些预定义的按钮，我们还可以自己绘制一些 BWCC 风格的按钮(见图 3-3)。绘制的规则见表 3-3。

表 3-3

按钮状态	VGA 中的位图 ID 值	EGA 中的位图 ID 值
标准	1 000+ButId	2 000+ButId
按下	3 000+ButId	4 000+ButId
焦点	5 000+ButId	6 000+ButId

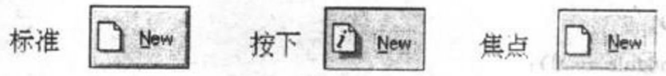


图 3-3　BWCC 风格按钮的位图

表 3-3 中的 ButId 是一个 1 000 以下的数，它是我们需要的按钮的 ID 值。比如三个位图的 ID 值(在 VGA 中)分别是 1 200、3 200 和 5 200，那么 ID 值等于 200 的按钮将根据不同的状态自动调用这三个位图(按钮与位图之间的联系就是这么默契)。

定义 BWCC 风格按钮的规则比较简单，而想要绘制出三个位图并且使它们具有动感和立体感就不那么容易了(你可以请美术功底好的朋友帮你的忙)。

3.3　检查框控件

TCheckBox 类是 TButton 类的派生类，图 3-4 所示为各种风格的检查框控件。其中封装了检查框控件的特性。在主窗口中可以用 TCheckBox 来创建一个检查框控件，也可以在对话框中使用 TCheckBox 对象，这样就可以方便地控制它。

用于窗口中的检查框控件构造函数是：

```
TCheckBox(TWindow * parent,
          int Id,
          const char far * title,
          int x, int y, int w, int h,
          TGroupBox * group = 0,
          TModule * module = 0);
```

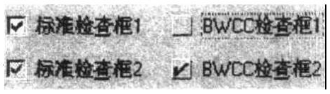


图 3-4 各种风格的检查框控件

它除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还增加了检查框上的字符串和检查框所在的组框(缺省时为 0)。

用于对话框的检查框控件构造函数是：

```
TCheckBox(TWindow * parent,
          int resourceId,
          TGroupBox * group = 0,
          TModule * module = 0);
```

它包含父窗口指针、资源 ID 值、所在组框以及一个模块指针，其它的参数由资源文件决定。

下面是两个典型的检查框控件对象构造语句：

* 用于窗口的典型的检查框构造语句：

new TCheckBox(this, ID_CHECKBOX, "检查框 1", 20, 20, 60, 30);

* 用于对话框的检查框构造语句：

new TCheckBox(this, ID_CHECKBOX);

一个检查框通常可以给用户提供双态选择，用户可以将控件设置为选择或非选择状态。在检查框组中，所有的检查框都可以被选择。比如，你可以使用检查框选择打开或改变应用程序的声音。除了双态检查框外，还有三态检查框，三态检查框增加了一种灰色状态。TCheckBox 类的成员函数使你可以轻松的控制检查框的状态。检查框也可以被包含在一个组框中。

当检查框出现在屏幕上时，它或者是选择状态或者是非选择状态，当用户点到一个检查框时，它发出一个消息，由检查框的父窗口来响应这个事件。决定一个检查框的状态通常应该是应用程序用户的任务而不是应用程序本身的任务，但是在有些情况下，你的应用程序需要控制检查框的状态。比如，当用户打开一个文本文件，你需要将检查框的文字自动改变为“以文本文件存盘”。TCheckBox 定义了一系列可以完成这样功能的成员函数：

成员函数	描述
Check 或 SetCheck(BF_CHECKED)	设置为选择状态
Uncheck 或 SetCheck(BF_UNCHECKED)	设置为非选择状态
Toggle	设置触发状态
SetState	设置高亮状态
SetStyle	改变按钮风格

3.4　Radio 按钮控件

TRadioButton 类代表窗口中相应的 Radio 按钮控件的界面元素(见图 3-5)。在父窗口中可以使用 TRadioButton 类来创建 Radio 按钮控件，也可以在对话框中使用 TRadioButton 对象，使得 Radio 按钮控件与应用程序更好的联系。

与检查框控件不同，Radio 按钮是从多个备选项中选择一个，比如，你可以用 Radio 按钮来选择应用程序中使用的声音种类，但只能选择其中的一种。

用于窗口中的 Radio 按钮控件构造函数是：

```
TRadioButton(TWindow * parent,
        int id,
        const char far * title,
        int x, int y, int w, int h,
        TGroupBox * group = 0,
        TModule * module = 0);
```

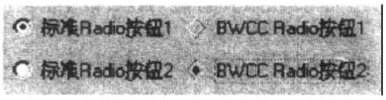


图 3-5　各种风格的 Radio 按钮控件

它除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还有 Radio 按钮上的字符串和 Radio 按钮所在的组框(缺省时为 0)。

用于对话框中的 Radio 按钮控件构造函数是：

```
TRadioButton(TWindow * parent,
        int resourceId,
        TGroupBox * group,
        TModule * module = 0);
```

它包含父窗口指针、资源 ID 值、所在组框以及一个模块指针，其它的参数由资源文件决定。

下面是两个典型的 Radio 按钮控件对象构造语句：

* 用于窗口的典型的 Radio 按钮构造语句：

```
new TRadioButton(this, ID_RADIOBUTTON, "Radio 按钮 1", 20, 20, 60, 30);
```

* 用于对话框的 Radio 按钮构造语句：

```
new TRadioButton(this, ID_RADIOBUTTON);
```

Radio 按钮有两种状态：选择和非选择状态。TRadioButton 类继承了它的基类 TCheckBox 的状态管理成员函数。另外，Radio 按钮可以是一个组框的项目，组框可视化地和逻辑地将其中的 Radio 控件归为一组。

因为 TRadioButton 是 TCheckBox 的派生类，所以 Radio 按钮拥有一些检查框的行为。当 Radio 按钮出现在屏幕上时，它或者是选择状态或者是非选择状态，当用户点到一个 Radio 按钮时，它发出一个消息，由 Radio 按钮的父窗口来响应这个事件。决定一个 Radio 按钮的状态通常应该是应用程序用户的任务而不是应用程序本身的任务，但是在有些情况下，你的应用程序需要控制 Radio 按钮的状态。TCheckBox 定义了一系列可以完成这样功能的成员函数(参考前面一节)。当你在 Radio 按钮中使用这些成员函数时，ObjectWindows 保证在一个组中只有一个 Radio 按钮被选择。

3.5 组 框 控 件

各种风格的组框控件见图 3-6 所示。

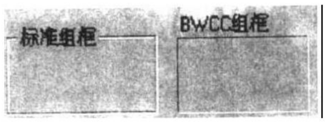


图 3-6 各种风格的组框控件

通常，用 TGroupBox 类来建立窗口中的一个组框。TGroupBox 的实例对象与组框界面元素相联系。表面上看组框在屏幕上并不作什么动作，它只是可视化地将一组选择框起来，比如，检查框、Radio 按钮和其它控件。其实组框扮演着一个非常重要的角色，它处理着组框中控件的状态变化(通常是 Radio 按钮)。

在最简单的方式中，组框定义一个包含标题的矩形区域，可视化地将其中的控件归为一组。用于窗口中的组框控件的构造函数是：

```
TGroupBox(TWindow *parent,
          int resourceId,
          const char far *text,
          int X, int Y, int W, int H,
          TModule *module = 0);
```

TGroupBox 类用于窗口的构造函数除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还增加了作为标记的文本字符串。

用于对话框中的组框控件的构造函数是：

```
TGroupBox(TWindow * Parent,
          int resourceId,
          TModule * module = 0);
```

TGroupBox 类用于对话框的构造函数除了包含父窗口指针、资源 ID 值以及一个模块指针，其他的参数由资源文件定义。

3.6 按钮控件程序实例(一)

在窗口中可以使用各种控件，但编写程序并不像在对话框那样容易。下面是在窗口中使用按钮、检查框和 Radio 按钮的例子(见图 3-7)，通过这个例子说明窗口中使用控件的方法。以后的控件程序例子都将基于对话框或用对话框作为客户程序的窗口。

在客户窗口的构造函数中定义了各种按钮和一个组框：

```
TMainWin::TMainWin()
  : TWindow(0, 0, 0)
{
  CheckBox = new TCheckBox(this, ID_CHECKBOX, "检查框", 20, 20, 80, 26, 0);
  new TButton(this, ID_BUTTON, "按钮", 120, 20, 64, 39, FALSE);
  GroupBox = new TGroupBox(this, ID_GROUPBOX, "组框", 20, 100, 176, 108);
```

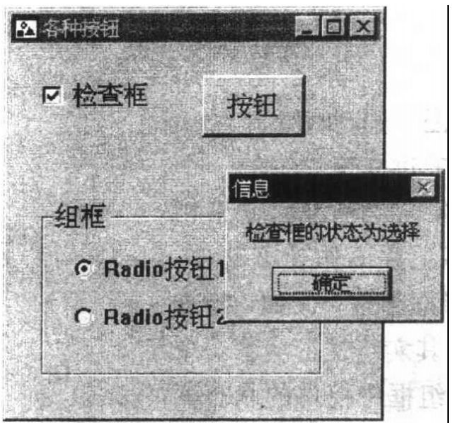


图 3-7　在窗口中使用各种按钮

```
    Radio1 = new TRadioButton(this, ID_RBUTTON1, "Radio 按钮 1",
                                            40, 130, 138, 24, GroupBox);
    Radio2 = new TRadioButton(this, ID_RBUTTON2, "Radio 按钮 2",
                                            40, 160, 138, 24, GroupBox);
}
```

客户窗口的事件响应表中定义了客户窗口需要响应的事件，其中包括按钮按下的事件、检查框点击的事件和组框事件：

```
DEFINE_RESPONSE_TABLE1(TMainWin, TWindow)
  EV_COMMAND(ID_BUTTON, OnButton),
  EV_COMMAND(ID_CHECKBOX, OnCheck),
  EV_CHILD_NOTIFY_ALL_CODES(ID_GROUPBOX, OnGroup),
END_RESPONSE_TABLE;
```

当按钮按下时通过函数 CheckBox->GetCheck() 查看检查框的状态；当组框中的控件被点击时通过函数 Radio1->GetCheck() 获得 Radio 按钮的状态。

下面列出程序代码。

```
/* Project button
      FILE:           button.h
***************************************
*/
#if !defined(__button_h)              // Sentry, use file only if it's not already included.
#define button_h
#include <owl\pch.h>

const WORD ID_BUTTON = 101;
const WORD ID_RBUTTON1 = 102;
const WORD ID_RBUTTON2 = 103;
const WORD ID_CHECKBOX = 104;
const WORD ID_GROUPBOX = 105;
```

```
class TMainWin : public TWindow
{
  public:
      TRadioButton * Radio1;
      TRadioButton * Radio2;
      TCheckBox *     CheckBox;
      TGroupBox *      GroupBox;

      TMainWin();

      // button handlers          //
      void OnButton();          // ID_BUTTON
      void OnCheck();        // ID_CHECKBOX
      void OnGroup(UINT);
  DECLARE_RESPONSE_TABLE(TMainWin);
};
#endif                                    // button_h sentry.

/* Project button
    FILE:          button.cpp
*************************************
*/
#include <owl\pch.h>
#include "button1.h"

DEFINE_RESPONSE_TABLE1(TMainWin, TWindow)
  EV_COMMAND(ID_BUTTON, OnButton),
  EV_COMMAND(ID_CHECKBOX, OnCheck),
  EV_CHILD_NOTIFY_ALL_CODES(ID_GROUPBOX, OnGroup),
END_RESPONSE_TABLE;

TMainWin::TMainWin()
  : TWindow(0, 0, 0)
{
  // Change the window's background color
  //
  SetBkgndColor(GetSysColor(COLOR_BTNFACE));

  CheckBox = new TCheckBox(this, ID_CHECKBOX, "检查框", 20, 20, 80, 26, 0);
  new TButton(this, ID_BUTTON, "按钮", 120, 20, 64, 39, FALSE);
  GroupBox = new TGroupBox(this, ID_GROUPBOX, "组框", 20, 100, 176, 108);
  Radio1 = new TRadioButton(this, ID_RBUTTON1, "Radio 按钮 1",40, 130, 138, 24, GroupBox);
  Radio2 = new TRadioButton(this, ID_RBUTTON2, "Radio 按钮 2",40, 160, 138, 24, GroupBox);
```

```
}

void
TMainWin::OnButton()
{
  if (CheckBox->GetCheck() == BF_CHECKED)
      MessageBox("检查框的状态为选择", "信息", MB_OK);
  else
      MessageBox("检查框的状态为非选择", "信息", MB_OK);
}

void
TMainWin::OnCheck()
{
  MessageBox("检查框被点击", "信息", MB_OK);
}

void
TMainWin::OnGroup(UINT /* notifyCode */)
{
  char textBuff[20];
  if (Radio1->GetCheck() == BF_CHECKED)
      Radio1->GetWindowText(textBuff, sizeof(textBuff));

  else
      Radio2->GetWindowText(textBuff, sizeof(textBuff));

  MessageBox(textBuff, "信息", MB_OK);
}

//-----------------------------------------

class TTestApp : public TApplication {
  public:
      TTestApp() : TApplication() {}
      void InitMainWindow();
};

void
TTestApp::InitMainWindow()
{
  MainWindow = new TFrameWindow(0, "各种按钮", new TMainWin);
}
```

```
int
OwlMain(int /* argc */, char* /* argv */ [])
{
  return TTestApp().Run();
}
```

3.7 按钮控件程序实例(二)

图 3-8 是在对话框中使用各种按钮的例子。例中使用 AppExpert 构造一个基于对话框的应用程序；使用 WorkShop 在对话框中加入一个三态检查框、一个组框、一个按钮和两个 Radio 按钮并设置检查框的状态为 Auto TriState；使用 ClassExpert 在对话框充当的客户窗口中定义了各种按钮。

下面得各种按钮的定义：

图 3-8 在对话框中使用各种按钮

```
TButton2DlgClient::TButton2DlgClient(TWindow* parent, TResId resId, TModule* module)
: TDialog(parent, resId, module)
{
//{{TButton2DlgClientXFER_USE}}
      Button = new TButton(this, IDC_BUTTON1);
      CheckBox = new TCheckBox(this, IDC_CHECKBOX1, 0);
      GroupBox = new TGroupBox(this, IDC_GROUPBOX1);
      Radio1 = new TRadioButton(this, IDC_RADIOBUTTON1, GroupBox);
      Radio2 = new TRadioButton(this, IDC_RADIOBUTTON2, GroupBox);

  SetTransferBuffer(&TButton2DlgClientData);
//{{TButton2DlgClientXFER_USE_END}}
}
```

当按钮按下时，获得检查框的状态。由于在资源编辑器中编辑的检查框为三态检查框，所以检查框的状态就有三种:选择、非选择和灰色。图中的检查框状态为灰色。

```
void TButton2DlgClient::OnButton()
{
  if (CheckBox->GetCheck() == BF_CHECKED)
      MessageBox("检查框的状态为选择", "信息", MB_OK);
  if (CheckBox->GetCheck() == BF_UNCHECKED)
      MessageBox("检查框的状态为非选择", "信息", MB_OK);
  if (CheckBox->GetCheck() == BF_GRAYED)
      MessageBox("检查框的状态为灰色", "信息", MB_OK);
}
```

分别响应两个 Radio 按钮按下的事件：

```
void TButton2DlgClient::OnRadio1()
{
  if (Radio1->GetCheck() == BF_CHECKED)
      MessageBox("Radio 按钮 1 的状态为选择", "信息", MB_OK);
  else
      MessageBox("Radio 按钮 1 的状态为非选择", "信息", MB_OK);
}

void TButton2DlgClient::OnRadio2()
{
  if (Radio2->GetCheck() == BF_CHECKED)
      MessageBox("Radio 按钮 2 的状态为选择", "信息", MB_OK);
  else
      MessageBox("Radio 按钮 2 的状态为非选择", "信息", MB_OK);
}
```

下面列出应用程序的代码，AppExpert 自动生成的代码省略，读者可以参看上一章。

```
//-----------------------------------------------------------------
//  FILE:          Btttdlgc.h
//  Class definition for TButton2DlgClient (TDialog).
//-----------------------------------------------------------------
#if !defined(bttn2dlc_h)          // Sentry, use file only if it's not already included.
#define bttn2dlc_h
#include "Bttn2App.rh"              // Definition of all resources.

#include <owl/Radiobut.h>
#include <owl/groupbox.h>
#include <owl/checkbox.h>
#include <owl/button.h>
//{{TDialog = TButton2DlgClient}}
struct TButton2DlgClientXfer {
//{{TButton2DlgClientXFER_DATA}}
    uint    CheckBox;
    uint    Radio1;
    uint    Radio2;
//{{TButton2DlgClientXFER_DATA_END}}
};

class TButton2DlgClient : public TDialog {
  public:
    TButton2DlgClient(TWindow* parent, TResId resId = IDD_CLIENT, TModule* module = 0);
    virtual ~TButton2DlgClient();
```

```
//{{TButton2DlgClientXFER_DEF}}
  protected:
    TButton *Button;
    TCheckBox *CheckBox;
    TGroupBox *GroupBox;
    TRadioButton *Radio1;
    TRadioButton *Radio2;
//{{TButton2DlgClientXFER_DEF_END}}
//{{TButton2DlgClientRSP_TBL_BEGIN}}
  protected:
  void OnButton();
  void OnCheck();
  void OnRadio1();
  void OnRadio2();
//{{TButton2DlgClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TButton2DlgClient);
};     //{{TButton2DlgClient}}
#endif     // bttn2dlc_h sentry.

/*   Project button
     FILE:          btttdlgc.cpp
     Source file for implementation of buttonTDLGClient (TDialog).
****************************************
//-------------------------------------------------------
//  FILE:          Btttdlgc.cpp
//  Source file for implementation of TButton2DlgClient (TDialog).
/-------------------------------------------------------
#include <owl/pch.h>
#include "Bttn2App.h"
#include "Bttn2DlC.h"


//
// Build a response table for all messages/commands handled by the application.
//
DEFINE_RESPONSE_TABLE1(TButton2DlgClient, TDialog)
//{{TButton2DlgClientRSP_TBL_BEGIN}}
  EV_BN_CLICKED(IDC_BUTTON1, OnButton),
  EV_BN_CLICKED(IDC_CHECKBOX1, OnCheck),
  EV_BN_CLICKED(IDC_RADIOBUTTON1, OnRadio1),
  EV_BN_CLICKED(IDC_RADIOBUTTON2, OnRadio2),
//{{TButton2DlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;
//{{TButton2DlgClient Implementation}}
```

```
//------------------------------------------
// TButton2DlgClient
// ~~~~~~~~
// Construction/Destruction handling.
//
static TButton2DlgClientXfer TButton2DlgClientData;
TButton2DlgClient::TButton2DlgClient(TWindow * parent, TResId resId, TModule * module)
: TDialog(parent, resId, module)
{
//{{TButton2DlgClientXFER_USE}}
    Button = new TButton(this, IDC_BUTTON1);
    CheckBox = new TCheckBox(this, IDC_CHECKBOX1, 0);
    GroupBox = new TGroupBox(this, IDC_GROUPBOX1);
    Radio1 = new TRadioButton(this, IDC_RADIOBUTTON1, GroupBox);
    Radio2 = new TRadioButton(this, IDC_RADIOBUTTON2, GroupBox);
  SetTransferBuffer(&TButton2DlgClientData);
//{{TButton2DlgClientXFER_USE_END}}
}

TButton2DlgClient::~TButton2DlgClient()
{
  Destroy();
}

void TButton2DlgClient::OnButton()
{
  if (CheckBox->GetCheck() == BF_CHECKED)
      MessageBox("检查框的状态为选择", "信息", MB_OK);
  if (CheckBox->GetCheck() == BF_UNCHECKED)
      MessageBox("检查框的状态为非选择", "信息", MB_OK);
  if (CheckBox->GetCheck() == BF_GRAYED)
      MessageBox("检查框的状态为灰色", "信息", MB_OK);
}

void TButton2DlgClient::OnCheck()
{
  MessageBox("检查框被点击", "信息", MB_OK);
}

void TButton2DlgClient::OnRadio1()
{
  if (Radio1->GetCheck() == BF_CHECKED)
      MessageBox("Radio 按钮 1 的状态为选择", "信息", MB_OK);
```

```
    else
        MessageBox("Radio 按钮 1 的状态为非选择", "信息", MB_OK);
}

void TButton2DlgClient::OnRadio2()
{
    if (Radio2->GetCheck() == BF_CHECKED)
        MessageBox("Radio 按钮 2 的状态为选择", "信息", MB_OK);
    else
        MessageBox("Radio 按钮 2 的状态为非选择", "信息", MB_OK);
}
```

3.8 静态控件

静态控件是一些通常不被用户修改而应用程序可以进行修改的文字或图形。因为用户的操作不直接作用于静态控件，所以应用程序就不需要接收控件消息。因此，－1 经常作为静态控件对象构造时的 ID 值，但如果你需要对静态控件进行操作时，它就需要一个唯一的 ID 值。

静态元素包括了用户不能改变的文本或图形，如果你想要在父窗口对象中创建静态控件来显示诸如 copyright 之类的文字，你就需要用到 TStatic 对象。TStatic 类也可以用来改变对话框对象中的静态控件文本。与其它控件类相同，在对话框中使用这个类可以使静态控件与应用程序方便地交流数据。

我们可以在资源编辑器中编辑一个对话框中的列表框，定义它的风格和 ID 值。静态控件的一个构造函数如下：

TStatic(TWindow * parent, int resourceId, const char far * title, int x, int y, int w, int h, UINT textLen = 0, TModule * module = 0);

这个构造函数是在窗口中使用静态对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、控件的坐标、字符串长度和一个模块指针)，其中有两个参数是控件特有的：控件显示的文本字符串和它的最大长度(包括最后的 NULL)。

静态控件的另外一个构造函数是：

```
TStatic(TWindow * parent,
            int resourceId,
            uint textLen = 0,
            TModule * module = 0);
```

这个构造函数是在对话框中使用静态对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、字符串长度和一个模块指针)，它的大小和位置都由资源文件决定。下面是两个典型的静态控件对象构造语句：

* 在窗口中使用静态控件

new TStatic(this, -1, "一个静态文本", 170, 20, 200, 24);

* 在对话框中使用静态控件

```
new TStatic(this, ID_STATIC);
```

如果你想改变静态控件的文本，你需要将控件对象作为父窗口对象的数据成员，这样你就可以调用静态控件对象的成员函数了。

TStatic 类改变了缺省的控件风格(缺省的控件风格为：WS_CHILD、WS_VISIBLE、WS_GROUP 和 WS_TABSTOP)，增加了 SS LEFT(左对齐方式)，删除了 WS_TABSTOP。要改变控件风格，需要修改 Attr.Style。下面的代码就是让静态控件的文本居中：

```
Attr.Style = (Attr.Style & ~SS_LEFT) | SS_CENTER;
```

应用程序改变静态控件的显示与改变编辑框控件的显示类似，而且静态控件的程序比较容易，本书不再给出例子，读者可以参考联机帮助和 BC 5 所附的例子(比如 Sysinfo 等)。

3.9 编辑框控件

图 3-9 是编辑框的例子。

编辑框控件是交互式的控件。在一个矩形区域中，用户可以填写、改变或删除文字。编辑框在用户输入数据时是非常有用的。编辑框提供用户文本输入，动态显示文本，用剪贴板 Clipboard 剪贴、拷贝和粘贴文字，多行输入等操作。

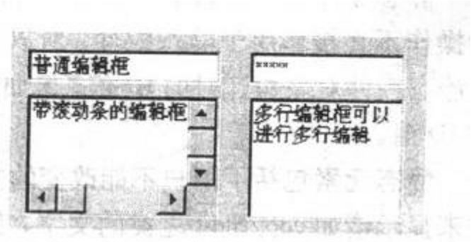


图 3-9　各种编辑框控件

TEdit 对象是编辑框控件界面元素的界面对象，可以在窗口和对话框中创建一个编辑框控件。在对话框中创建 TEdit 对象有利于增加编辑框控件与应用程序的联系。大多数 TEdit 的成员函数管理着编辑框控件的文本，TEdit 还具有一些成员函数，它们自动响应编辑框控件的父窗口的菜单项选择，包括 cut、copy、和 paste。

有两种风格的编辑框控件：单行和多行编辑框控件。多行编辑框控件允许编辑多行文本。

在资源编辑器中可以编辑对话框中的编辑框控件。编辑框控件的一个构造函数如下：

```
TEdit(TWindow * parent,
        int Id,
        const char far * text,
        int x, int y, int w, int h,
        uint textLen = 0,
        bool multiline = false,
        TModule * module = 0);
```

这个构造函数是在窗口中使用编辑对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、编辑框中的字符串、控件的坐标、字符串长度、多行编辑或单行编辑开关和一个模块指针)。

编辑框控件的另外一个构造函数是：

```
TEdit(TWindow * parent,
```

```
        int resourceID,
        uint textLen = 0,
        TModule * module = 0);
```

这个构造函数是在对话框中使用编辑框对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、字符串长度和一个模块指针)，其他参数都由资源文件决定。

下面是两个典型的编辑框控件对象构造语句：

* 在窗口中使用编辑框控件

```
new TEdit(this, ID_EDIT, "编辑框控件", 20,20,200,25, 10, false);
```

* 在对话框中使用编辑框控件

```
new TEdit(this, ID_EDIT,10);
```

TEditSearch 是一个响应菜单项命令 Find、Replace 和 FindNext 的编辑框控件。

TEditFile 是一个文件编辑窗口。TEditFile 的数据成员和成员函数管理着文件对话框，并自动响应文件命令，比如，Open、Read、Write、Save 和 SaveAs。

3.10 编辑框控件程序实例

本小节是一个在对话框中使用编辑框的例子，通过例子讨论下面三点：

1. 普通的单行编辑框和多行编辑框的使用方法；
2. 特殊编辑框的使用方法；
3. 对编辑框中的输入串进行检查，并给出警告信息框。

首先，我们使用 AppExpert 生成一个基于对话框的应用程序框架；使用 WorkShop 在对话框中加入编辑框，其中口令编辑框属性设置为 PassWord；在 ClassExpert 的帮助下定义编辑框对象。在客户窗口的构造函数中构造编辑框，在析构函数中不需要显式地删除这些编辑框对象，因为它们是客户窗口的子窗口，在客户窗口删除时将自动删除全部的子窗口。

```
static TEditDlgClientXfer TEditDlgClientData;
TEditDlgClient::TEditDlgClient(TWindow * parent, TResId resId, TModule * module)
:  TDialog(parent, resId, module)
{
//{{TEditDlgClientXFER_USE}}
    edit1 = new TEdit(this, IDC_EDIT1, 255);
    edit2 = new TEdit(this, IDC_EDIT2, 255);
    edit3 = new TEdit(this, IDC_EDIT3, 255);
    edit4 = new TEdit(this, IDC_EDIT4, 255);
    edit5 = new TEdit(this, IDC_EDIT5, 255);
    edit6 = new TEdit(this, IDC_EDIT6, 255);
    edit7 = new TEdit(this, IDC_EDIT7, 255);

  SetTransferBuffer(&TEditDlgClientData);
//{{TEditDlgClientXFER_USE_END}}
}
```

数据缓冲区设置为：

```
struct TEditDlgClientXfer {
//{{TEditDlgClientXFER_DATA}}
    char    edit1[ 255 ];
    char    edit2[ 255 ];
    char    edit3[ 255 ];
    char    edit4[ 255 ];
    char    edit5[ 255 ];
    char    edit6[ 255 ];
    char    edit7[ 255 ];
//{{TEditDlgClientXFER_DATA_END}}
};
```

此时编译并执行程序，可发现程序中各个编辑框都可以工作，其中口令输入框还可以显示星号代替输入的字符。其实，对于那些不打算对编辑框控件进行直接操作应用程序，我们不必在对话框窗口中构造对应的编辑框对象。

我们在客户窗口的事件响应表中定义一系列编辑框事件的响应方法，其中包括编辑框中的内容被修改、失去焦点、获得焦点以及用户输入的字符串达到最大长度等等。当需要随时监测或修改用户的输入时，这些事件响应就非常有用。

```
DEFINE_RESPONSE_TABLE1(TEditDlgClient, TDialog)
//{{TEditDlgClientRSP_TBL_BEGIN}}
  EV_EN_CHANGE(IDC_EDIT3, ENChange),
  EV_EN_KILLFOCUS(IDC_EDIT3, ENKillfocus),
  EV_EN_MAXTEXT(IDC_EDIT3, ENMaxtext),
//{{TEditDlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;
```

相应的事件处理函数如下：

```
void TEditDlgClient::ENChange()
{}

void TEditDlgClient::ENKillfocus()
{
MessageBox("编辑框失去焦点", "信息", MB_OK);
}

void TEditDlgClient::ENMaxtext()
{
MessageBox("编辑框中的字符串达到最大长度", "信息", MB_OK);
}
```

对编辑框中的输入格式进行检查是非常重要的，通过它可以检查出用户输入是否正确，增加程序的容错能力，使你的程序变得更加安全易用。我们可以规定一个特定的编辑

框用来输入一定范围内的数字，另一个编辑框可以输入一些特定的字符串。只使用编辑框类很难完成这些功能(你需要随时检查用户的输入并进行修改或提示)，为此，我们使用了 TValidator 类和它的子类来定义一个规定数据格式的格式字符串。使用这个类需要加入头文件：

```
#include <owl/validate.h>
```

对于普通的编辑框对象，典型的构造函数是：

```
Edit1 = new TEdit(this, ID_EDIT1, "", 20,50,200,25, 10, false);
```

对于只能输入数字的编辑框对象，我们使用了 TFilterValidator 类，其中的"0—9"指定了用户输入字符的范围。

```
Edit2 = new TEdit(this, ID_EDIT2, "", 20,110,200,25, 10, false);
Edit2->SetValidator(new TFilterValidator("0-9"));
```

对于只能输入英文字母的编辑框对象，我们同样使用了 TFilterValidator 类，其中的"A—Za—z"指定了用户输入字符的范围。

```
Edit3 = new TEdit(this, ID_EDIT3, "", 20,170,200,25, 10, false);
Edit3->SetValidator(new TFilterValidator("A-Za-z. "));
```

对于只能输入规定的字符串的编辑框对象，我们使用了 TPXPictureValidator 类，它的构造函数中的"学士，硕士，博士，啥都不是"字符串说明了用户可以输入的选择，TRUE 说明了应用程序可以自动判断用户的可能输入并以此修改编辑框控件中的字符串。比如我们键入“啥”的时候，编辑框中就会自动出现“啥都不是”。

```
Edit4 = new TEdit(this, ID_EDIT4, "", 20,230,200,25, 10, false);
Edit4->SetValidator(new TPXPictureValidator("学士,硕士,博士,啥都不是", TRUE));
```

当我们只想要用户在编辑框中编辑一定范围的数字时，我们就要用到 TRangeValidator 类，它的构造函数中的两个数字就说明了用户可以编辑的最小和最大的数字。如果用户的输入不对，它会给出信息。

```
Edit5 = new TEdit(this, ID_EDIT5, "", 20,290,200,25, 10, false);
Edit5->SetValidator(new TRangeValidator(1897, 1997));
```

对于应用程序的特殊要求，我们可以提供继承 TValidator 类或它的子类来实现对用户输入的实时监测。

使用 ClassExpert，重载了对话框的 SetupWindow 函数，我们知道，它是对话框构造中调用的最后一个函数：

```
void TEditDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  edit4->SetValidator(new TFilterValidator("0-9"));
  edit5->SetValidator(new TFilterValidator("A-Za-z. "));
  edit6->SetValidator(new TPXPictureValidator("学士,硕士,博士,啥都不是", TRUE));
  edit7->SetValidator(new TRangeValidator(1897, 1997));
}
```

对于中文应用程序，为了给出中文提示信息，需要重新编辑资源中的一些字符串：

```
// Validator messages (include/owl/validate.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_VALPXPCONFORM      "编辑框需要的格式是:\n""%s"""
  IDS_VALINVALIDCHAR     "输入有误，请检查!"
  IDS_VALNOTINRANGE      "数字输入的范围是 %ld 到 %ld."
  IDS_VALNOTINLIST       "输入不满足要求!"
END
```

图 3-10、图 3-11 分别给出了编辑框应用程序的执行情况及其项目文件。

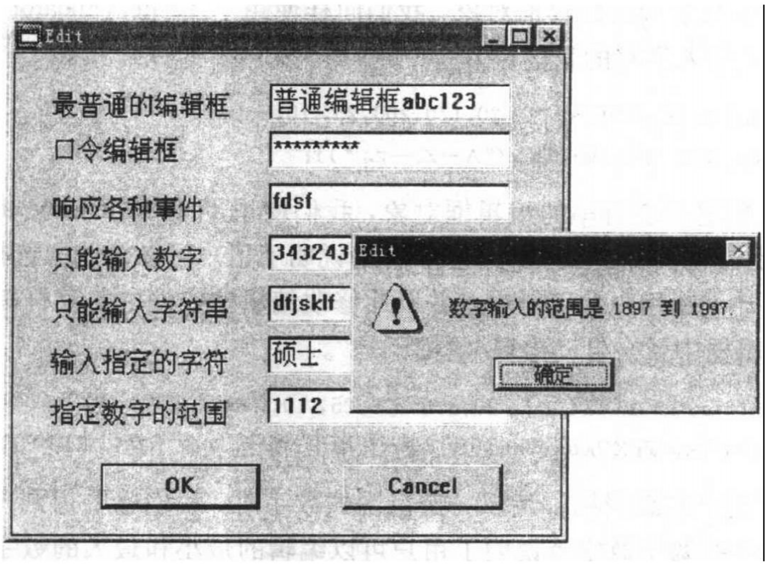


图 3-10　编辑框程序执行情况

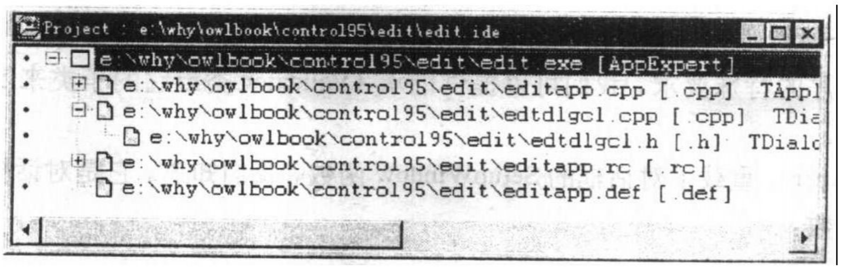


图 3-11　编辑框程序的项目文件

下面是部分程序代码：

```
//------------------------------------------------------------------------
//  FILE:         EdtDlgCl.h
//  Class definition for TEditDlgClient (TDialog).
//------------------------------------------------------------------------
#if !defined(edtdlgcl_h)              // Sentry, use file only if it's not already included.
```

```
#define edtdlgcl_h
#include "EditApp.rh"                    // Definition of all resources.

#include <owl/edit.h>
//{{TDialog = TEditDlgClient}}
struct TEditDlgClientXfer {
//{{TEditDlgClientXFER_DATA}}
    char    edit1[ 255 ];
    char    edit2[ 255 ];
    char    edit3[ 255 ];
    char    edit4[ 255 ];
    char    edit5[ 255 ];
    char    edit6[ 255 ];
    char    edit7[ 255 ];
//{{TEditDlgClientXFER_DATA_END}}
};

class TEditDlgClient : public TDialog {
  public:
    TEditDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TEditDlgClient();

//{{TEditDlgClientXFER_DEF}}
  protected:
    TEdit *edit1;
    TEdit *edit2;
    TEdit *edit3;
    TEdit *edit4;
    TEdit *edit5;
    TEdit *edit6;
    TEdit *edit7;

//{{TEditDlgClientXFER_DEF_END}}

//{{TEditDlgClientVIRTUAL_BEGIN}}
  public:
    virtual void SetupWindow();
//{{TEditDlgClientVIRTUAL_END}}

//{{TEditDlgClientRSP_TBL_BEGIN}}
  protected:
  void ENChange();
  void ENKillfocus();
```

```
    void ENMaxtext();
  //{{TEditDlgClientRSP_TBL_END}}
  DECLARE_RESPONSE_TABLE(TEditDlgClient);
  };    //{{TEditDlgClient}}

  #endif     // edtdlgcl_h sentry.

  //----------------------------------------------------------------
  //  FILE:            EdtDlgCl.cpp
  //  Source file for implementation of TEditDlgClient (TDialog).
  //----------------------------------------------------------------

  #include <owl/pch.h>
  #include <owl/validate.h>
  #include "EditApp.h"
  #include "EdtDlgCl.h"

  //
  // Build a response table for all messages/commands handled by the application.
  //
  DEFINE_RESPONSE_TABLE1(TEditDlgClient, TDialog)
  //{{TEditDlgClientRSP_TBL_BEGIN}}
    EV_EN_CHANGE(IDC_EDIT3, ENChange),
    EV_EN_KILLFOCUS(IDC_EDIT3, ENKillfocus),
    EV_EN_MAXTEXT(IDC_EDIT3, ENMaxtext),
  //{{TEditDlgClientRSP_TBL_END}}
  END_RESPONSE_TABLE;

  //{{TEditDlgClient Implementation}}

  //----------------------------------------------------------------
  // TEditDlgClient
  // ~~~~~~~~
  // Construction/Destruction handling.
  //
  static TEditDlgClientXfer TEditDlgClientData;

  TEditDlgClient::TEditDlgClient(TWindow* parent, TResId resId, TModule* module)
  :
    TDialog(parent, resId, module)
  {
  //{{TEditDlgClientXFER_USE}}
      edit1 = new TEdit(this, IDC_EDIT1, 255);
```

```
    edit2 = new TEdit(this, IDC_EDIT2, 255);
    edit3 = new TEdit(this, IDC_EDIT3, 255);
    edit4 = new TEdit(this, IDC_EDIT4, 255);
    edit5 = new TEdit(this, IDC_EDIT5, 255);
    edit6 = new TEdit(this, IDC_EDIT6, 255);
    edit7 = new TEdit(this, IDC_EDIT7, 255);

  SetTransferBuffer(&TEditDlgClientData);
}

TEditDlgClient::~TEditDlgClient()
{
  Destroy();
}

void TEditDlgClient::ENChange()
{
  // INSERT>> Your code here.
}

void TEditDlgClient::ENKillfocus()
{
  // INSERT>> Your code here.
  MessageBox("编辑框失去焦点", "信息", MB_OK);
}

void TEditDlgClient::ENMaxtext()
{
  // INSERT>> Your code here.
  MessageBox("编辑框中的字符串达到最大长度", "信息", MB_OK);
}

void TEditDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  edit4->SetValidator(new TFilterValidator("0-9"));
  edit5->SetValidator(new TFilterValidator("A-Za-z. "));
  edit6->SetValidator(new TPXPictureValidator("学士,硕士,博士,啥都不是", TRUE));
  edit7->SetValidator(new TRangeValidator(1897, 1997));
}
```

3.11 列表框控件

图 3-12 是各种风格的列表框的例子。

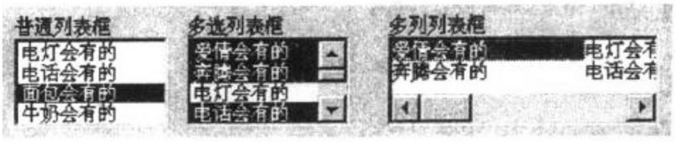


图 3-12 各种风格的列表框

在窗口对象中创建一个列表框控件的时候需要使用 TListBox 类。TListBox 类代表一个与之相关的列表框控件界面元素的界面对象，与其他控件类相同，在对话框中使用这个类可以使列表框控件与应用程序方便地交流数据。

使用列表框可以使用户方便地从列表中选择一些项目，TListBox 类封装了列表框的行为。TListBox 定义了四种用途的成员函数：

创建列表框；

改变列表框中的项目；

查询列表框中的项目；

获得用户的选择。

列表框控件的一个构造函数如下：

```
TListBox(TWindow * Parent,
          int Id,
          int x, int y, int w, int h,
          TModule * module = 0);
```

这个构造函数是在窗口中使用列表框对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、控件的坐标和一个模块指针)。

列表框控件的另外一个构造函数是：

```
TListBox(TWindow * Parent,
          int resourceId,
          TModule * module = 0)
```

这个构造函数是在对话框中使用列表框对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值和一个模块指针)，它的大小和位置都由资源文件决定。

下面是两个典型的列表框控件对象构造语句：

* 在窗口中使用列表框控件

```
ListBox1 = new TListBox(this, IDC_LISTBOX1, 10,20,50,30);
```

* 在对话框中使用列表框控件

```
ListBox1 = new TListBox(this, IDC_LISTBOX1);
```

我们可以在资源编辑器中编辑一个对话框中的列表框，定义它的风格和 ID 值。

TListBoxData 结构用来传输列表框控件的内容。因为列表框需要传输一系列值(包括字

符串、项目数据和选择索引)，所以传输缓冲区 TListBoxData 类用一系列数据成员来保存列表框的信息(见表 3-4)。

表 3-4

数据成员	类 型	描 述
ItemDatas	TUint32Array *	列表框中的每一相用 uint32 变量保存
SelIndices	TIntArray *	在多选框中保存被选择的项目索引
Strings	TStringArray *	保存列表框中全部的字符串

TListBoxData 提供表 3-5 中所列的成员函数用于操作控件。

表 3-5

成员函数	描 述
AddItemData	在 ItemDatas 数组中增加项目数据
AddString	在 Strings 数组中增加字符串，并可任意选择它
AddStringItem	在 Strings 数组中增加字符串，并可任意选择它；在 ItemDatas 数组中增加项目数据
GetSelString	按给出的索引获得选择的字符串
GetSelStringLength	返回所给索引中选择的字符串长度
ResetSelections	从 SelIndices 数组中删除所有的选项
Select	选择索引中的字符串
SelectString	选择所给的字符串

3.12 列表框控件程序实例

下面是一个列表框的使用例子，它演示三种风格的列表框(普通列表框、多选列表框和多列列表框)。首先，使用 AppExpert 生成一个基于对话框的应用程序框架，使用 WorkShop 在对话框中加入三个不同风格的列表框对象，它们的大小、位置和风格等属性都由资源文件中的具有相同资源 ID 值的列表框确定。普通列表框的风格为 Single，多选列表框的风格为 MultiPle，多列列表框的风格为 Multi Column。然后，在 ClassExpert 中增加三个列表框的对象变量和数据缓冲区。

三个列表框的对象变量定义如下：

```
static TListDlgClientXfer TListDlgClientData;
TListDlgClient::TListDlgClient(TWindow * parent, TResId resId, TModule * module)
:  TDialog(parent, resId, module)
{
//{{TListDlgClientXFER_USE}}
    list1 = new TListBox(this, IDC_LISTBOX1);
    list2 = new TListBox(this, IDC_LISTBOX2);
```

```
      list3 = new TListBox(this, IDC_LISTBOX3);
    SetTransferBuffer(&TListDlgClientData);
  //{{TListDlgClientXFER_USE_END}}
  }
```

数据缓冲区的定义如下：

```
struct TListDlgClientXfer {
//{{TListDlgClientXFER_DATA}}
    TListBoxData   list1;
    TListBoxData   list2;
    TListBoxData   list3;
//{{TListDlgClientXFER_DATA_END}}
};
```

使用 ClassExpert 重载客户窗口的 SetupWindow 函数，给每一个列表框控件增加了选项。

```
void TListDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  list1->AddString("面包会有的");
                 ...
  list2->AddString("面包会有的");
                 ...
  list3 ->AddString("面包会有的");
                 ...
}
```

在客户窗口的析构函数中并没有显式地删除列表框对象，这是因为三个列表框对象都是客户窗口的子窗口，所以当客户窗口被删除时，作为子窗口的列表框将被自动删除。

图 3-13，图 3-14 分别给出了列表框应用程序的执行情况及其项目文件。

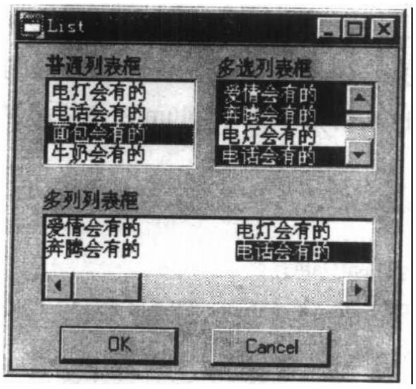


图 3-13　列表框控件例子

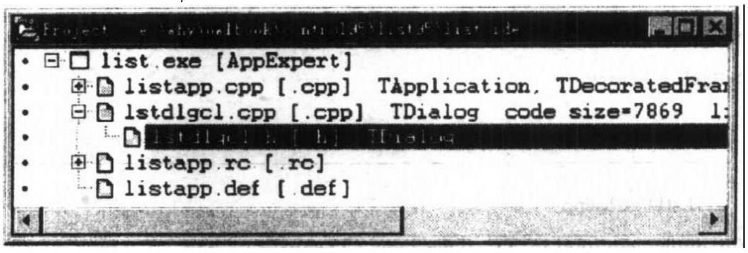


图 3-14 列表框应用程序的项目文件

下面是列表框应用程序的部分代码：

```
//----------------------------------------------------------------
//  FILE:            LstDlgCl.h
//  Class definition for TListDlgClient (TDialog).
//----------------------------------------------------------------
#if ! defined(lstdlgcl_h)          // Sentry, use file only if it's not already included.
#define lstdlgcl_h
#include "ListApp.rh"                      // Definition of all resources.
#include <owl/listbox.h>

//{{TDialog = TListDlgClient}}
struct TListDlgClientXfer {
//{{TListDlgClientXFER_DATA}}
    TListBoxData  list1;
    TListBoxData  list2;
    TListBoxData  list3;
//{{TListDlgClientXFER_DATA_END}}
};

class TListDlgClient : public TDialog {
  public:
    TListDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TListDlgClient();
//{{TListDlgClientXFER_DEF}}
  protected:
    TListBox  * list1;
    TListBox  * list2;
    TListBox  * list3;

//{{TListDlgClientXFER_DEF_END}}

//{{TListDlgClientVIRTUAL_BEGIN}}
  public:
    virtual void SetupWindow();
```

```
//{{TListDlgClientVIRTUAL_END}}
};    //{{TListDlgClient}}
#endif    // lstdlgcl_h sentry.

//----------------------------------------
//  FILE:        LstDlgCl.cpp
//  Source file for implementation of TListDlgClient (TDialog).
/----------------------------------------
#include <owl/pch.h>
#include "ListApp.h"
#include "LstDlgCl.h"

//{{TListDlgClient Implementation}}
//----------------------------------
// TListDlgClient
// ~~~~~~~~
// Construction/Destruction handling.
//
static TListDlgClientXfer TListDlgClientData;

TListDlgClient::TListDlgClient(TWindow * parent, TResId resId, TModule * module)
:  TDialog(parent, resId, module)
{
//{{TListDlgClientXFER_USE}}
    list1 = new TListBox(this, IDC_LISTBOX1);
    list2 = new TListBox(this, IDC_LISTBOX2);
    list3 = new TListBox(this, IDC_LISTBOX3);
  SetTransferBuffer(&TListDlgClientData);
//{{TListDlgClientXFER_USE_END}}
}

TListDlgClient::~TListDlgClient()
{
  Destroy();
}

void TListDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  list1->AddString("面包会有的");
  list1->AddString("牛奶会有的");
  list1->AddString("电灯会有的");
  list1->AddString("电话会有的");
```

```
    list2->AddString("面包会有的");
    list2->AddString("牛奶会有的");
    list2->AddString("电灯会有的");
    list2->AddString("电话会有的");
    list2->AddString("奔腾会有的");
    list2->AddString("爱情会有的");
    list2->AddString("晴天会有的");
    list2->AddString("一切都会有的");

    list3 ->AddString("面包会有的");
    list3 ->AddString("牛奶会有的");
    list3 ->AddString("电灯会有的");
    list3 ->AddString("电话会有的");
    list3 ->AddString("奔腾会有的");
    list3 ->AddString("爱情会有的");
    list3 ->AddString("晴天会有的");
    list3 ->AddString("一切都会有的");
}
```

3.13 组合框控件

图 3-15 是各种风格的组合框的例子。如果要在窗口对象中增加一个组合框就需要使用 TComboBox 类。

TComboBox 类是 TListBox 类的派生类，它继承了一系列父类的行为。与其它控件对象相同，在对话框对象中使用 TComboBox 对象可以使组合框与应用程序的联系更加简单。

使用组合框可以使用户方便地从表中选择项目，TComboBox 类封装了组合框的行为。

组合框控件的一个构造函数如下：

```
TComboBox(TWindow * parent,
          int id,
          int x, int y, int w, int h,
          uint32 style,
          uint textLen,
          TModule * module = 0);
```

图 3-15 各种组合框

这个构造函数是在窗口中使用组合框对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、控件的坐标、风格参数、字符串长度和一个模块指针)。其中风格参数的取值可以是：CBS_SIMPLE，CBS_DROPDOWN，CBS_DROPDOWNLIST，CBS_OWNERDRAWFIXED，或 CBS_OWNERDRAWVARIABLE。

组合框控件的另外一个构造函数是：

```
TComboBox(TWindow * parent,
```

```
int ResourceId,
uint textLen = 0,
TModule * module = 0);
```

这个构造函数是在对话框中使用组合框对象所需要的，它包括一系列参数(父窗口指针、资源 ID 值、字符串长度和一个模块指针)，其它参数都由资源文件决定。

下面是两个典型的组合框控件对象构造语句：

* 在窗口中使用组合框控件

```
ComboBox1 = new TComboBox(this, IDC_COMBOBOX1, 10,20,50,30,
            CBS_DROPDOWNLIST, 20);
```

* 在对话框中使用组合框控件

```
ComboBox2 = new TComboBox(this, IDC_COMBOBOX2, 20);
```

我们可以在资源编辑器中编辑一个对话框中的组合框，定义它的风格和 ID 值。

有三种组合框可以选择，它们是由不同的风格参数决定的(见表 3-6)。

表 3-6

组合框类型	风格参数
简单组合框	CBS_SIMPLE
下拉编辑组合框	CBS_DROPDOWN
下拉静态组合框	CBS_DROPDOWNLIST

组合框组合了两种不同的控件:列表框和编辑框控件(或静态控件)。它提供了与列表框相似的功能，让用户从可滚动的表中用鼠标选择一项。在编辑组合框中，编辑框在列表框的上面，提供了另外一种选择的机制，允许用户直接键入文本。TComboBox 类派生于 TListBox 类，继承了修改、查询和选择项目等成员函数，另外 TComboBox 增加了下拉列表的机制。

TComboBoxData 类负责传输 TComboBox 类数据到一个传输缓冲区的界面对象。组合框需要传输一系列信息(包括选项数据、选择的项目和选择项目的索引)。组合框的传输缓冲区 TComboBoxData 包含了表 3-7 中所列的数据成员用于保存组合框的信息。

表 3-7

数据成员	类型	描　　述
ItemDatas	TUint32Array *	列表框中的每一相用 uint32 变量保存
Selection	char *	保存所选项的字符串
Strings	TStringArray *	保存列表框中全部的字符串

TComboBoxData 提供表 3-8 中所列的成员函数用于操作控件。

表 3-8

成 员	函 数 描 述
AddString	在 Strings 数组中增加字符串并可任意选择它
AddStringItem	在 Strings 数组中增加字符串并可任意选择它；在 ItemDatas 数组中增加项目数据
Clear	清除所有数据
GetItemDatas	返回 ItemDatas 的一个参数
GetSelCount	返回选择项目的数字
GetSelection	返回当前选择的参数
GetSelIndex	返回当前选择的索引
GetSelString	将当前选择的字符串拷贝到一个字符串缓冲区中
GetSelStringLength	返回当前选择的字符串长度
GetStrings	返回在输入区的字符串
ResetSelections	将当前选项设置为空字符串，并设置索引为 CB_ERR
Select	选择索引中的字符串
SelectString	选择所给的字符串

3.14 组合框控件程序实例

下面是一个组合框的使用例子，它演示三种风格的组合框(普通组合框、下拉编辑组合框和下拉静态组合框)。首先，使用 AppExpert 生成一个基于对话框的应用程序框架，使用 WorkShop 在对话框中加入三个不同风格的组合框对象，它们的大小、位置和风格等属性都由资源文件中的具有相同资源 ID 值的组合框确定：普通组合框的风格为 Simple，下拉编辑组合框的风格为 Dropdown，下拉静态组合框的风格为 Dropdowm List。然后，在 ClassExpert 中增加三个组合框的对象变量和数据缓冲区。

三个组合框对象变量的定义如下：

```
static TComboDlgClientXfer TComboDlgClientData;

TComboDlgClient::TComboDlgClient(TWindow * parent, TResId resId, TModule * module)
: TDialog(parent, resId, module)
{
//{{TComboDlgClientXFER_USE}}
    ComboBox1 = new TComboBox(this, IDC_COMBOBOX1, 1);
    ComboBox2 = new TComboBox(this, IDC_COMBOBOX2, 1);
    ComboBox3 = new TComboBox(this, IDC_COMBOBOX3, 1);
  SetTransferBuffer(&TComboDlgClientData);
//{{TComboDlgClientXFER_USE_END}}
```

```
}
```

数据缓冲区的定义如下：

```
struct TComboDlgClientXfer {
//{{TComboDlgClientXFER_DATA}}
    TComboBoxData ComboBox1;
    TComboBoxData ComboBox2;
    TComboBoxData ComboBox3;
//{{TComboDlgClientXFER_DATA_END}}
};
```

使用 ClassExpert 重载客户窗口的 SetupWindow 函数，给每一个列表框控件增加了选项。

```
void TComboDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  ComboBox1->AddString("面包会有的");
                ...
  ComboBox2->AddString("面包会有的");
                ...
  ComboBox3 ->AddString("面包会有的");
                ...
;
```

在客户窗口的析构函数中并没有显式地删除组合框对象，这是因为三个组合框对象都是客户窗口的子窗口，所以当客户窗口被删除时，作为子窗口的组合框将被自动删除。

图 3-16，图 3-17 分别给出了组合框应用程序的执行情况及其项目文件。

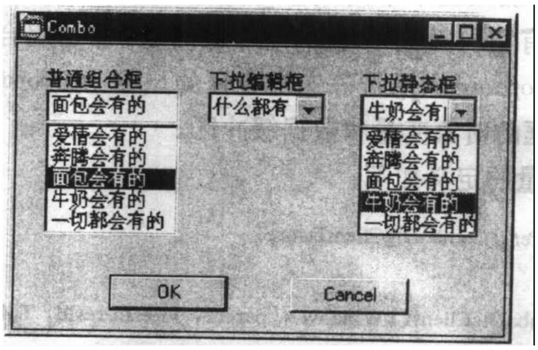


图 3-16 运行中的组合框例子

下面是组框应用程序中的部分代码：

```
//-----------------------------------------------
// FILE:          CmbDlgCl.h
// Class definition for TComboDlgClient (TDialog).
//-----------------------------------------------
#if ! defined(cmbdlgcl_h)          // Sentry, use file only if it's not already included.
```

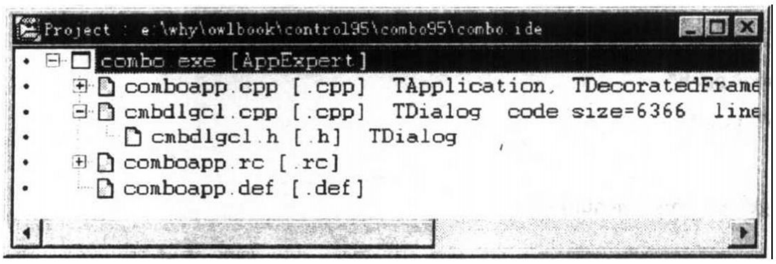


图 3-17 组合框应用程序项目文件

```
#define cmbdlgcl_h
#include "ComboApp.rh"                    // Definition of all resources.
#include <owl/combobox.h>

//{{TDialog = TComboDlgClient}}
struct TComboDlgClientXfer {
//{{TComboDlgClientXFER_DATA}}
    TComboBoxData ComboBox1;
    TComboBoxData ComboBox2;
    TComboBoxData ComboBox3;
//{{TComboDlgClientXFER_DATA_END}}
};

class TComboDlgClient : public TDialog {
  public:
    TComboDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TComboDlgClient();

//{{TComboDlgClientXFER_DEF}}
  protected:
    TComboBox  * ComboBox1;
    TComboBox  * ComboBox2;
    TComboBox  * ComboBox3;
//{{TComboDlgClientXFER_DEF_END}}
//{{TComboDlgClientVIRTUAL_BEGIN}}
  public:
    virtual void SetupWindow();
//{{TComboDlgClientVIRTUAL_END}}
};    //{{TComboDlgClient}}
#endif      // cmbdlgcl_h sentry.

//------------------------------------------
//  FILE:          CmbDlgCl.cpp
//  Source file for implementation of TComboDlgClient (TDialog).
```

```
//---------------------------------------
#include <owl/pch.h>
#include "ComboApp.h"
#include "CmbDlgCl.h"

//{{TComboDlgClient Implementation}}
//------------------------------------
// TComboDlgClient
// ~~~~~~~~~
// Construction/Destruction handling.
//
static TComboDlgClientXfer TComboDlgClientData;

TComboDlgClient::TComboDlgClient(TWindow * parent, TResId resId, TModule * module)
: TDialog(parent, resId, module)
{
//{{TComboDlgClientXFER_USE}}
    ComboBox1 = new TComboBox(this, IDC_COMBOBOX1, 1);
    ComboBox2 = new TComboBox(this, IDC_COMBOBOX2, 1);
    ComboBox3 = new TComboBox(this, IDC_COMBOBOX3, 1);
  SetTransferBuffer(&TComboDlgClientData);
//{{TComboDlgClientXFER_USE_END}}
}

TComboDlgClient::~TComboDlgClient()
{
  Destroy();
}

void TComboDlgClient::SetupWindow()
{
  TDialog::SetupWindow();

  ComboBox1->AddString("面包会有的");
  ComboBox1->AddString("牛奶会有的");
  ComboBox1->AddString("奔腾会有的");
  ComboBox1->AddString("爱情会有的");
  ComboBox1->AddString("一切都会有的");

  ComboBox2->AddString("面包会有的");
  ComboBox2->AddString("牛奶会有的");
  ComboBox2->AddString("奔腾会有的");
  ComboBox2->AddString("爱情会有的");
```

```
    ComboBox2->AddString("一切都会有的");

    ComboBox3->AddString("面包会有的");
    ComboBox3->AddString("牛奶会有的");
    ComboBox3->AddString("奔腾会有的");
    ComboBox3->AddString("爱情会有的");
    ComboBox3->AddString("一切都会有的");
}
```

3.15 滚动条控件

滚动条是一种常用的控件，用于改变用户对窗口、列表框或组合框的视觉范围。有时，你也可能需要一个单独的滚动条来完成某些特殊的任务，比如在温度控制器中控制温度显示或改变绘图颜色等，这时你可以使用 TScrollBar 对象。

TScrollBar 对象描述了横向和纵向的滚动条控件(见图 3-18)。大多数的 TScrollBar 成员函数管理滚动条的指示位置的位置条，控制它的位置和范围。TScrollBar 类的一个比较特殊的地方是它的一系列成员函数是通过响应广播消息自动的调节位置条的。所以不要在具有窗口滚动条的窗口中使用 TScrollBar。

用于窗口中的滚动条控件构造函数是：

```
TScrollBar(TWindow * parent,
        int id,
        int x, int y, int w, int h,
        bool isHScrollBar,
        TModule * module = 0);
```

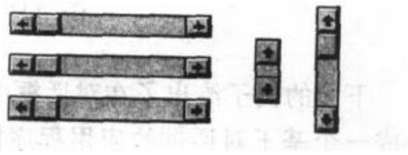


图 3-18　横向和纵向的滚动条

它除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还增加了滚动条的方向(横向还是纵向)。

用于对话框中的滚动条控件构造函数是：

```
TScrollBar(TWindow * parent,
        int resourceId,
        TModule * module = 0);
```

它包含父窗口指针、资源 ID 值以及一个模块指针，其它的参数由资源文件决定。

下面是两个典型的滚动条控件对象构造语句：

* 用于窗口的典型的滚动条构造语句：

new TScrollBar(this, ID_SCROLL1, 20,20,100,20, true)

* 用于对话框的滚动条构造语句：

new TScrollBar(this, ID_SCROLL1)

TScrollBarData 结构包含了表示滚动条上的位置条范围值。TScrollBar 类的成员函数 GetRange 调用 TScrollBarData 来获得在滚动条中位置条的最大值和最小值。GetPosition 调用 TScrollBarData 来获得滚动条中当前位置条的位置。

你可能需要滚动条处理自己的消息，为此，TWindow 类提供了内建的分配滚动条消息到滚动条的功能，TWindow::EvHScroll 函数或 TWindow::EvVScroll 函数根据情况执行 TScrollBar 的成员函数。你可以派生出新的滚动条类，但应该保证首先调用原来的成员函数，这些成员函数使滚动条正确地更新到新的位置上。表 3-9 是滚动条消息与滚动条成员函数的对照。

表 3-9

滚动条消息	滚动条成员函数
SB_LINEUP	SBLineUp
SB_LINEDOWN	SBLineDown
SB_PAGEUP	SBPageUp
SB_PAGEDOWN	SBPageDown
SB_THUMBPOSITION	SBThumbPosition
SB_THUMBTRACK	SBThumbTrack
SB_TOP	SBTop
SB_BOTTOM	SBBottom

3.16 滚动条控件程序实例

下面的例子给出了在对话框中滚动条(见图 3-19)的使用方法。首先，使用 AppExpert 生成一个基于对话框的应用程序框架，使用 WorkShop 在对话框中加入不同风格的滚动条对象，同时定义了两个编辑框控件用来显示滚动条的位置，它们的大小、位置和风格等属性都由资源文件中的具有相同资源 ID 值的组合框确定。横向滚动条的风格为 Horizontal，纵向滚动条的风格为 Vertical。编辑框设置为 Disable(用户无法改变)。

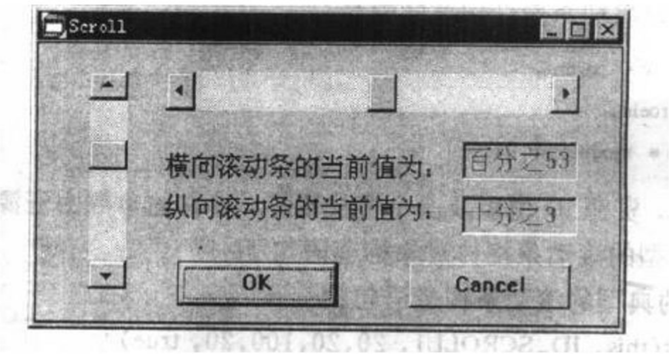


图 3-19　在对话框中使用滚动条

在 ClassExpert 中增加三个组合框的对象变量和数据缓冲区：

```
static TScrollDlgClientXfer TScrollDlgClientData;

TScrollDlgClient::TScrollDlgClient(TWindow * parent, TResId resId, TModule * module)
```

```
: TDialog(parent, resId, module)
{
//{{TScrollDlgClientXFER_USE}}
    ScrollBar1 = new TScrollBar(this, IDC_SCROLLBAR1);
    ScrollBar2 = new TScrollBar(this, IDC_SCROLLBAR2);
    Edit1 = new TEdit(this, IDC_EDIT1, 255);
    Edit2 = new TEdit(this, IDC_EDIT2, 255);
  SetTransferBuffer(&TScrollDlgClientData);
//{{TScrollDlgClientXFER_USE_END}}
}
```

数据缓冲区的定义为：

```
struct TScrollDlgClientXfer
{
//{{TScrollDlgClientXFER_DATA}}
    TScrollDlgClientXfer();
    TScrollBarData     ScrollBar1;
    TScrollBarData     ScrollBar2;
    char     Edit1[ 255 ];
    char     Edit2[ 255 ];
//{{TScrollDlgClientXFER_DATA_END}}
};
```

使用 ClassExpert 在对话框的事件响应表中定义了一系列滚动条的事件响应，其中包括滚动条向上、向下的滚动事件。

```
DEFINE_RESPONSE_TABLE1(TScrollDlgClient, TDialog)
//{{TScrollDlgClientRSP_TBL_BEGIN}}
      EV_SB_LINEUP(IDC_SCROLLBAR1, EVScoll1),
      EV_SB_PAGEUP(IDC_SCROLLBAR1, EVScoll1),
      EV_SB_LINEDOWN(IDC_SCROLLBAR1, EVScoll1),
      EV_SB_PAGEDOWN(IDC_SCROLLBAR1, EVScoll1),
      EV_SB_THUMBPOSITION(IDC_SCROLLBAR1, EVScoll1),

      EV_SB_LINEUP(IDC_SCROLLBAR2, EVScoll2),
      EV_SB_PAGEUP(IDC_SCROLLBAR2, EVScoll2),
      EV_SB_LINEDOWN(IDC_SCROLLBAR2, EVScoll2),
      EV_SB_PAGEDOWN(IDC_SCROLLBAR2, EVScoll2),
      EV_SB_THUMBPOSITION(IDC_SCROLLBAR2, EVScoll2),
//{{TScrollDlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;
```

定义缓冲区的构造函数，设置滚动条的值。

```
TScrollDlgClientXfer::TScrollDlgClientXfer()
{
```

```
    ScrollBar1.LowValue = 0;
    ScrollBar1.HighValue = 100;
    ScrollBar1.Position = 50;

    ScrollBar2.LowValue = 0;
    ScrollBar2.HighValue = 10;
    ScrollBar2.Position = 5;
}
```

当位置条的位置改变时就改变静态控件的显示，调用函数 GetPosition()来获得滚动按钮的位置。

```
char str[10];
sprintf(str, "百分之%d",ScrollBar1->GetPosition());
Edit1->SetText(str);
```

图 3-20 所示为滚动条应用程序的项目文件。

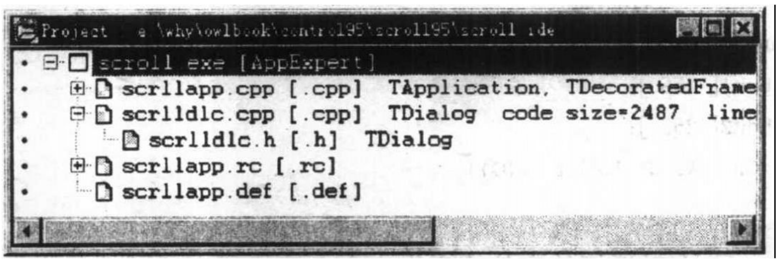


图 3-20　滚动条应用程序的项目文件

下面是滚动条应用程序的部分代码：

```
//----------------------------------------------------------------
//  FILE:          ScrllDlC.h
//  Class definition for TScrollDlgClient (TDialog).
//----------------------------------------------------------------
#if !defined(scrlldlc_h)              // Sentry, use file only if it's not already included.
#define scrlldlc_h

#include "ScrllApp.rh"                // Definition of all resources.
#include <owl/edit.h>
#include <owl/scrollba.h>

//{{TDialog = TScrollDlgClient}}
struct TScrollDlgClientXfer
{
//{{TScrollDlgClientXFER_DATA}}
    TScrollDlgClientXfer();
    TScrollBarData    ScrollBar1;
    TScrollBarData    ScrollBar2;
```

```
      char      Edit1[ 255 ];
      char      Edit2[ 255 ];
  //{{TScrollDlgClientXFER_DATA_END}}
  };


  class TScrollDlgClient : public TDialog {
    public:
      TScrollDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
      virtual ~TScrollDlgClient();

  //{{TScrollDlgClientXFER_DEF}}
    protected:
      TScrollBar  * ScrollBar1;
      TScrollBar  * ScrollBar2;
      TEdit  * Edit1;
      TEdit  * Edit2;

  //{{TScrollDlgClientXFER_DEF_END}}

  //{{TScrollDlgClientRSP_TBL_BEGIN}}
    protected:
    void EVScoll1();
    void EVScoll2();
  //{{TScrollDlgClientRSP_TBL_END}}
  DECLARE_RESPONSE_TABLE(TScrollDlgClient);
  };      //{{TScrollDlgClient}}
  #endif      // scrlldlc_h sentry.


  //----------------------------------------------------------
  //  FILE:           ScrllDlC.cpp
  //  Source file for implementation of TScrollDlgClient (TDialog).
  //----------------------------------------------------------
  #include <owl/pch.h>
  #include <stdio.h>
  #include "ScrllApp.h"
  #include "ScrllDlC.h"


  //
  // Build a response table for all messages/commands handled by the application.
  //
  DEFINE_RESPONSE_TABLE1(TScrollDlgClient, TDialog)
  //{{TScrollDlgClientRSP_TBL_BEGIN}}
        EV_SB_LINEUP(IDC_SCROLLBAR1, EVScoll1),
```

```
        EV_SB_PAGEUP(IDC_SCROLLBAR1, EVScoll1),
        EV_SB_LINEDOWN(IDC_SCROLLBAR1, EVScoll1),
        EV_SB_PAGEDOWN(IDC_SCROLLBAR1, EVScoll1),
        EV_SB_THUMBPOSITION(IDC_SCROLLBAR1, EVScoll1),

        EV_SB_LINEUP(IDC_SCROLLBAR2, EVScoll2),
        EV_SB_PAGEUP(IDC_SCROLLBAR2, EVScoll2),
        EV_SB_LINEDOWN(IDC_SCROLLBAR2, EVScoll2),
        EV_SB_PAGEDOWN(IDC_SCROLLBAR2, EVScoll2),
        EV_SB_THUMBPOSITION(IDC_SCROLLBAR2, EVScoll2),
//{{TScrollDlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{TScrollDlgClient Implementation}}

TScrollDlgClientXfer::TScrollDlgClientXfer()
{
  ScrollBar1.LowValue = 0;
  ScrollBar1.HighValue = 100;
  ScrollBar1.Position = 50;

  ScrollBar2.LowValue = 0;
  ScrollBar2.HighValue = 10;
  ScrollBar2.Position = 5;
}

//-----------------------------------------------------------
// TScrollDlgClient
// ~~~~~~~~~
// Construction/Destruction handling.
//
static TScrollDlgClientXfer TScrollDlgClientData;

TScrollDlgClient::TScrollDlgClient(TWindow * parent, TResId resId, TModule * module)
:  TDialog(parent, resId, module)
{
//{{TScrollDlgClientXFER_USE}}
    ScrollBar1 = new TScrollBar(this, IDC_SCROLLBAR1);
    ScrollBar2 = new TScrollBar(this, IDC_SCROLLBAR2);
    Edit1 = new TEdit(this, IDC_EDIT1, 255);
    Edit2 = new TEdit(this, IDC_EDIT2, 255);

  SetTransferBuffer(&TScrollDlgClientData);
```

```
//{{TScrollDlgClientXFER_USE_END}}
}

TScrollDlgClient::~TScrollDlgClient()
{
  Destroy();
}

void TScrollDlgClient::EVScoll1()
{
        char str[10];
        sprintf(str, "百分之%d",ScrollBar1->GetPosition());
        Edit1->SetText(str);
}

void TScrollDlgClient::EVScoll2()
{
        char str[10];
        sprintf(str, "十分之%d",ScrollBar2->GetPosition());
        Edit2->SetText(str);
}
```

3.17 Slider 控件

Slider 控件是一种特殊的滚动条控件(见图 3-21)，抽象类 TSlider 派生于 TScrollBar 类中。与普通的滚动条一样，Slider 具有最大和最小的位置，可以通过鼠标和键盘来移动指示指针，键盘的 Home 和 End 键使指示指针移动到最大和最小的位置上。你可以通过 TSlider 的成员函数来设置指示指针的位置，设置指针移动一步的距离。

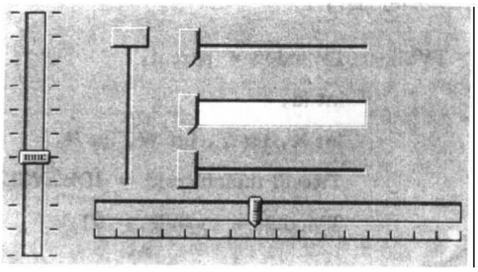


图 3-21 各种 Slider 控件

与其它控件的构造函数一样，TSlider 类的一个构造函数除了包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针)以外，还增加了 TResId 对象，它包含了一个位图的资源 ID 值。这个位图显示了 Slider 控件中的指针样子。

下面是 TSlider 的构造函数：

```
THSlider(TWindow * parent,
         int id,
         int X, int Y, int W, int H,
         TResId thumbResId = IDB_HSLIDERTHUMB,
```

```
TModule * module = 0);
```

TSlider 类中包括了一系列纯虚函数，表 3-10 列出了 TSlider 中的纯虚函数。

表 3-10

函数名	功 能
HitTest	确定一个点是否在指示指针中或是 Slider 的其他热点区域中
NotifyParent	通知 Slider 父窗口一个滚动事件
PaintRuler	绘制标尺
PaintSlot	绘制指示指针所在的槽
PointToPos	将一个 Slider 中的点转变为 Slider 的位置
PosToPoint	将 Slider 的位置转变为 Slider 中的一个点
SetupThumbRgn	设置指针区域，缺省时，是一个简单的区域
SlideThumb	滚动指示指针

从 TSlider 类中派生出两个类，THSlider 类和 TVSlider 类。它们是水平的和垂直的 Slider 版本。THSlider 和 TVSlider 都只有一个构造函数，且类似于 TSlider 的构造函数，不同的是它们都有一个指示指针的缺省位图。

横向控件：

```
THSlider(TWindow * parent,
        int id,
        int X, int Y, int W, int H,
        TResId thumbResId = IDB_HSLIDERTHUMB,
        TModule * module = 0);
```

纵向控件：

```
TVSlider(TWindow * parent,
        int id,
        int X, int Y, int W, int H,
        TResId thumbResId = IDB_VSLIDERTHUMB,
        TModule * module = 0);
```

3.18 Gauge 控件

Gauge 是一种特殊控件(见图 3-22)，它可以表示一个正在进行的过程的进度。TGauge 类封装了 Gauge 控件的特性，它派生于 TControl 类。Gauge 对象通过横向或纵向的条状矩形显示了一个过程的进展程度，我们经常可以在安装软件中看到这类控件。一般，我们用横向的控件来表示一个过程的状态，用纵向的 Gauge 控件来表示信息分析的结果。

TGauge 类的构造函数如下：

```
TGauge(TWindow * parent,
```

```
const char far * title,
int id,
int X, int Y, int W, int H,
bool isHorizontal = true,
int margin = 0,
TModule * module = 0);
```

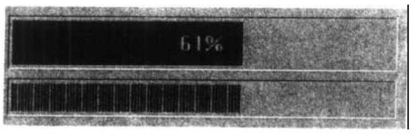


图 3-22 各种 Gauge 控件

TGauge 类的构造函数包含通常控件类的参数(父窗口指针、资源 ID 值、控件的坐标和大小以及一个模块指针),isHorizontal 参数决定你使用的是一个水平的 Gauge 控件还是一个垂直的 Gauge 控件。如果 isHorizontal 是 true,Gauge 控件将水平显示;如果 isHorizontal 是 false,Gauge 控件将垂直显示,缺省时是水平控件。margin 参数决定 Gauge 控件的边际大小。

3.19 Gauge 控件程序实例

下面的例子使用了 Gauge 控件进行过程监测,它显示了一个软件的安装过程的进度(见图 3-23)。

在客户窗口的构造函数中构造了一个 Gauge 控件的对象。"%d%%"定义了在 Gauge 控件中显示过程的进度百分比;TRUE 说明 Gauge 控件是横向的。

```
GaugeTDLGClient::GaugeTDLGClient(TWindow * parent, TResId resId, TModule * module)
      : TDialog(parent, resId, module)
{
  Position=0;
  SetupGauge = new TGauge(this, ID_SOLIDGAUGE,
40, 71, 300, 30, TRUE, 2);
}
```

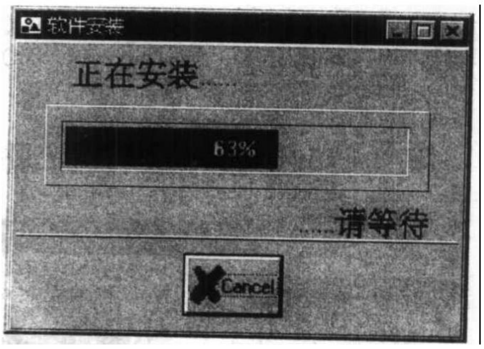


图 3-23 软件的安装过程

事件响应表中定义了客户窗口所响应的事件:时钟事件。例子中的 Gauge 控件对象的变化是用时钟事件驱动的。

```
DEFINE_RESPONSE_TABLE1(GaugeTDLGClient, TDialog)
//{{GaugeTDLGClientRSP_TBL_BEGIN}}
    EV_WM_TIMER,
//{{GaugeTDLGClientRSP_TBL_END}}
END_RESPONSE_TABLE;
```

在 SetupWindow 函数中申请了一个时钟,并设置了 Gauge 控件对象的范围。

```
void GaugeTDLGClient::SetupWindow ()
{
      TDialog::SetupWindow();
```

```
        SetTimer(TIME_SETUP,200);
        SetupGauge->SetRange(0, 100);
}
```

由于使用了时钟，所以在客户窗口的析构函数中删除指定的时钟。

```
GaugeTDLGClient::~GaugeTDLGClient ()
{
        Destroy();
        KillTimer(TIME_SETUP);
}
```

EvTimer 函数是窗口类中的标准时钟事件处理函数，如果你的应用程序中有若干个时钟，你就需要在函数中判断是哪个时钟发出的事件。SetupGauge->SetValue(Position)函数将 Gauge 控件对象的指示设置到所给的值。

```
void GaugeTDLGClient::EvTimer (uint timerId)
{
        TDialog::EvTimer(timerId);
        if(Position>=100) Position=0;
        Position++;
        SetupGauge->SetValue(Position);
}
```

图 3-24 所示为 Gauge 控件应用程序的项目文件情况。

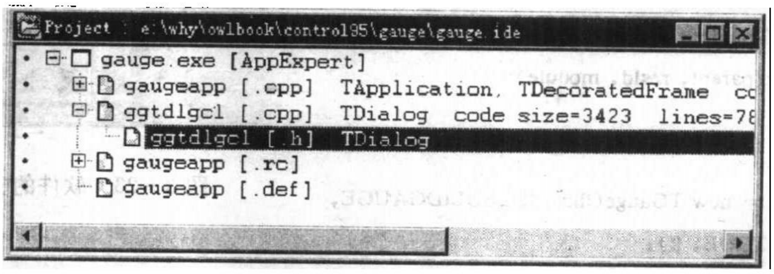


图 3-24　Gauge 控件应用程序项目文件

下面是 Gauge 应用程序的部分代码：

```
/*****************************************************
    FILE:          ggtdlgcl.h
    Class definition for GaugeTDLGClient (TDialog).
*****************************************************/
#if !defined(__ggtdlgcl_h)          // Sentry, use file only if it's not already included.
#define ggtdlgcl_h

#include <owl\pch.h>
#include "Gaugeapp.rh"                           // Definition of all resources.

//{{TDialog = GaugeTDLGClient}}
```

```
class GaugeTDLGClient : public TDialog {
public:
    int Position;
    TGauge * SetupGauge;
    GaugeTDLGClient (TWindow *parent, TResId resId = IDD_CLIENT, TModule *module = 0);
    virtual ~GaugeTDLGClient ();

//{{GaugeTDLGClientVIRTUAL_BEGIN}}
public:
    virtual void SetupWindow ();
//{{GaugeTDLGClientVIRTUAL_END}}

//{{GaugeTDLGClientRSP_TBL_BEGIN}}
protected:
    void EvTimer (uint timerId);
//{{GaugeTDLGClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(GaugeTDLGClient);
};    //{{GaugeTDLGClient}}
#endif                                  // ggtdlgcl_h sentry.

/*************************************
    FILE:          ggtdlgcl.cpp
    Source file for implementation of GaugeTDLGClient (TDialog).
*************************************/
#include <owl\pch.h>
#include "Gaugeapp.h"
#include "ggtdlgcl.h"

#define TIME_SETUP        100
#define ID_SOLIDGAUGE      200
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(GaugeTDLGClient, TDialog)
//{{GaugeTDLGClientRSP_TBL_BEGIN}}
    EV_WM_TIMER,
//{{GaugeTDLGClientRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{GaugeTDLGClient Implementation}}

////////////////////////////////////////
```

```
// GaugeTDLGClient
// =========
// Construction/Destruction handling.
GaugeTDLGClient::GaugeTDLGClient (TWindow *parent, TResId resId, TModule *module)
    : TDialog(parent, resId, module)
{
        Position=0;
        SetupGauge = new TGauge(this, "%d%%", ID_SOLIDGAUGE, 30, 50, 220, 30, TRUE, 2);
}

GaugeTDLGClient::~GaugeTDLGClient ()
{
        Destroy();
        KillTimer(TIME_SETUP);
}

void GaugeTDLGClient::SetupWindow ()
{
        TDialog::SetupWindow();
        SetTimer(TIME_SETUP,200);
        SetupGauge->SetRange(0, 100);
}

void GaugeTDLGClient::EvTimer (uint timerId)
{
        TDialog::EvTimer(timerId);
        if(Position>=100) Position=0;
        Position++;
        SetupGauge->SetValue(Position);
}
```

第 4 章

对话框类

对话框类封装了具有对话框行为的界面对象。TDialog 类提供了所有类型对话框的初始化、创建和执行。就像你可以从 TWindow 类中派生出特殊的窗口类一样，你也可以从 TDialog 类中派生出你的应用程序所需要的对话框类。ObjectWindows 体系也提供了一系列 Windows 常用的对话框类型，比如选择文件名、字体、颜色等通用对话框。

4.1　对　话　框

对话框对象的使用与窗口对象的使用非常相像，对于简单的临时对话框，你可以用父窗口的成员函数来控制。通过调用成员函数，你可以构造、执行和关闭一个对话框对象。使用对话框需要遵循下列的步骤：

1. 构造对话框对象；
2. 执行对话框对象；
3. 关闭并析构对话框对象。

对话框可以用来作为应用程序的主窗口。

4.2　各种风格的对话框

由于对话是常用的信息输入输出方式，所以对话框的风格和样式就显得很重要。OWL 中提供了对话框 3-D 界面，如果你想要使用 Microsoft 公司的 CTL3D DLL，只要在应用程序中调用 TApplication::EnableCtl3d 即可(在 Windows95 中不需要)，ObjectWindows 将自动发送 WM_CTLCOLOR 消息到 CTL3D DLL；ObjectWindows 也为对话框提供了 BWCC 界面。一个好的对话框经常使软件开发者自豪，使用户赞叹。下面我们来看看经常使用的一些对话框风格：

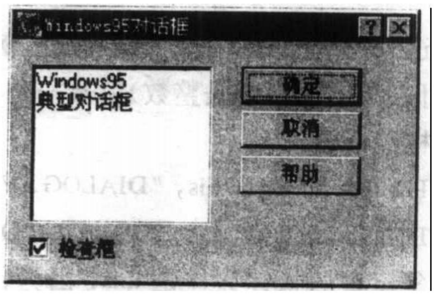


图 4-1　Windows95 中的典型对话框

* 首先是最通常的对话框(见图 4-1)，它具有一些普通的控件。
* Borland 风格的对话框相当美观(见图 4-2)，对话框中特有的灰色条纹使对话框有

一种雕刻的质感，再配上 BWCC 风格的控件更显得富丽堂皇。在 Windows3.1 中，BWCC 风格的对话框确实可以使应用程序增色不少。

* Mircosoft 提供的一种三维风格的对话框，它使对话框的底色变为浅灰色，同时对话框中的控件都很具有立体感，图中显示的编辑框和检查框就是典型的立体风格。为了鱼和熊掌兼得，可以在 MS 3D 风格的对话框中加入了 BWCC 控件，形成了图 4-3 中的情况。

图 4-2　Windows3.1 下具有 BWCC 风格的对话框

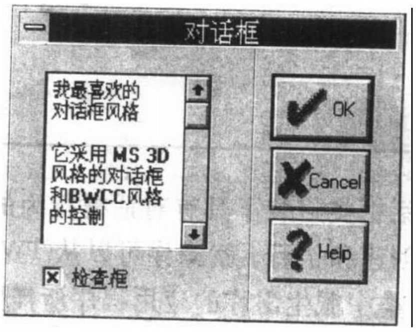


图 4-3　具有 MS 3D 风格的对话框

4.3　构造对话框对象

对话框是通过对话框资源设计和创建的，你可以使用 Borland 公司的 Resource Workshop 资源编辑器方便地创建对话框资源并连接到应用程序中。对话框资源描述了对话框和其上的控件对象的样子和位置，包括 buttons、list boxes 和 group boxes 等，但它不包括对话框的响应行为。

每一个对话框资源都有一个资源 ID 值使它能够和特定的对话框对象联系起来，ID 值可以是一个字符串或整数。

使用对话框的构造函数：

```
TDialog(TWindow * parent,
        TResId resId,
        TModule * module = 0);
```

这个构造函数构造了一个对话框对象，它的参数包括一个父窗口的指针、一个资源 ID 值(可以是字符串或是整数)以及一个模块指针(缺省时为 0)。下面是两个典型的对话框对象的构造语句：

```
TDialog dialog1(this, "DIALOG_1");
TDialog dialog2(this, IDD_DIALOG);
```

父窗口的指针往往是 this，因为你通常是在窗口对象的成员函数中创建对话框对象。如果你不在窗口对象中创建对话框对象，就用应用程序的主窗口作为父窗口，因为在 ObjectWindows 应用程序中，主窗口总是其他窗口的父窗口。例如：

```
TDialog myDialog(GetApplication()->GetMainWindow(), IDD_DLG);
```

当你用对话框对象作为 TFrameWindow 类或 TFrameWindow 的派生类对象的客户窗口

时，情况就有一些特殊，构造函数把对话框对象传输给 TFrameWindow::Init 函数，Init 函数自动设置对话框的父窗口。

4.4　执行一个对话框对象

执行一个对话框对象与创建并显示一个窗口有些相似，但是因为对话框对象总是作为临时窗口显示，故一些步骤可以省略，那些步骤可以省略，则取决于你怎么使用它。通常可以以模式对话框和无模式对话框的方式执行对话框。

模式对话框

大多数对话框是模式对话框，当模式对话框被显示时，用户不能选择或使用它的父窗口，用户必须在关闭了对话框后才能继续运行。

模式对话框用 TDialog::Execute 函数来执行。当用户关闭对话框后，函数返回一个整数值来表明用户是怎么关闭对话框的。返回值是用户按下的控件的 ID 值，比如 IDOK 是 OK 按钮的 ID 值；IDCANCEL 是 Cancel 按钮的 ID 值。如果对话框对象是动态创建的，记着将它删除。

下面的例子假设你已经有一个对话框资源，ID 值是 IDD_MYDIALOG，对话框上有两个按钮 OK 和 Cancel：

```
if(TDialog(this, IDD_MYDIALOG).Execute() == IDOK)
  // 用户点击了 OK 按钮
else
  // 用户点击了 CANCEL 按钮
```

当超出局部变量的活动范围时，对话框对象被删除，但是对话框资源却不被删除。你可以用一个对话框资源创建和删除任意多的对话框对象。

无模式对话框

与模式对话框不同，无模式对话框允许在对话框打开期间使用应用程序的其他窗口。使用无模式对话框可以让使用者在打开对话框时继续其他操作或获得信息等。

调用 TDialog::Create 来执行一个无模式对话框。执行时，你必须显式地说明对话框的可见性，你应该在资源风格中设置标志 WS_VISIBLE 或是调用 ShowWindow 函数强行让对话框显示。

下面的例子告诉你资源的大致描述方式：

```
DIALOG_1 DIALOG 10, 10, 140, 40
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog 1"
{
  PUSHBUTTON "按钮", IDOK, 58, 23, 25, 16
}
```

现在，你可以用下面的代码创建无模式对话框

```
TDialog dialog1(this, "DIALOG_1");
dialog1.Create();
```

这个对话框还没有显示在屏幕上，你必须保证对话框的风格中设置了 WS_VISIBLE 标志，例如：

STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE

或者在调用 Create 函数后增加 ShowWindow 函数：

```
TDialog dialog1(this, "DIALOG_1");
dialog1.Create();
dialog1.ShowWindow(SW_SHOW);
```

TDialog::CmOk 函数和 TDialog::CmCancel 函数关闭对话框并删除对话框对象。这些函数在 TDialog 类的事件响应表中定义，用来处理来自 OK 和 Cancel 按钮的 IDOK 和 IDCANCEL 消息。CmOk 函数并调用 CloseWindow 来关闭无模式对话框；CmCancel 函数传递 IDCANCEL 参数并调用 Destroy 函数。它们都可以关闭对话框。如果你重载了 CmOk 和 CmCancel 函数，你应该调用它们的原函数否则你需要自己进行关闭和清理工作。另外，你可以在对话框的父窗口的构造函数中创建对话框对象，这样，你只创建了对话框对象一次，以后对对话框的任何变动比如位置、活动焦点等都将在下次对话框打开时保存原样。

就像其它的子窗口一样，对话框对象在父窗口撤销时自动删除，这样，如果关闭了对话框的父窗口，对话框就自动删除了。

有一点比较特殊，ObjectWindows 应用程序中，无模式对话框被用户关闭后，它的指针并不被删除，也不会调用对话框类的析构函数。如果对话框对象有一个父窗口，当父窗口撤销时对话框的析构函数被调用并删除对话框对象指针；如果它没有父窗口，指针被自动删除。你可以通过在构造函数中设置 wfDeleteOnClose 标志重载这个操作。设置 wfDeleteOnClose 标志使得 ObjectWindows 在接收 WM_CLOSE 后自动删除无模式对话框的指针并调用析构函数，但设置这个标志对模式对话框没有作用。

下面的代码中，父窗口的构造函数只是构造了一个对话框对象，其他的函数执行对话框的创建和显示：

```
class TParentWindow : public TFrameWindow
{
  public:
    TParentWindow(TWindow * parent, const char * title);
    void CmDOIT();
  protected:
    TDialog  *dialog;
};
  ...
void TParentWindow::CmDOIT()
{
  dialog = new TDialog(this, IDD_EMPLOYEE_INFO);
  dialog->Create();
}
```

4.5 关闭对话框

每一个对话框必须有一种用户关闭方式，通常按 OK 或 Cancel 按钮可以关闭它。TDialog 类通过 CmOk 和 CmCancel 函数来响应这些事件。CmOk 调用 CloseWindow 函数，CloseWindow 函数又调用 CanClose 函数看是否关闭对话框，如果 CanClose 函数返回 true，CloseWindow 函数将传输对话框数据并调用 TWindow::CloseWindow 函数来关闭它；CmCancel 调用 EvClose 函数，EvClose 函数又调用 Destroy 函数关闭对话框，这种方式不检验是否可以关闭，也不传输数据。

4.6 在资源编辑器中编辑对话框

对话框中包含了一些控件，它们是作为对话框的子窗口(比如检查框、编辑框和滚动条等等)，这些控件通常是用作应用程序与用户的信息交换的界面。根据应用程序的要求，我们在一个空的对话框中加入特定的控件，使用 Borland C++中的 Resource Workshop(资源编辑器)可以使这一过程变得简单。

在资源编辑器中编辑对话框通常需要四个步骤：

1. 启动对话框编辑器；
2. 创建一个新的对话框(或编辑一个已经存在的对话框)；
3. 测试对话框；
4. 将包含对话框的资源文件存盘。

4.7 用对话框作主窗口

想要把对话框作为主窗口，最好用一个框架窗口作主窗口而对话框作为主窗口的客户窗口。首先，从 TApplication 类中派生出一个应用程序类，构造函数不变，只改动 InitMainWindow 函数。在新的 InitMainWindow 函数中构造一个框架窗口对象，将对话框作为客户窗口。

下面的代码实现这样的功能：

```
#include <owl\applicat.h>

class TMyApp : public TApplication
{
  public:
    TMyApp() : TApplication("对话框作主窗口") {}
    void InitMainWindow();
};
```

```
void TMyApp::InitMainWindow()
{
  SetMainWindow(new TFrameWindow(0, "一个对话框",
                               new TDialog(0, "MYDIALOG"), true));
}
int OwlMain(int argc, char * argv[])
{
  return TMyApp().Run();
}
```

TFrameWindow 的构造函数为作为客户窗口的对话框打开 autocreation 功能，而不管你自己设置的状态。

使用对话框作为主窗口时应注意：

1. 在删除对话框对象的时候不能破坏框架窗口，必须显式地删除框架窗口；

2. 不能在对话框中动态地直接增加资源，因为实际上它不是主窗口，你应该在框架窗口中增加资源(比如，你想在对话框中使用 SetIcon 函数来增加应该图标，你就必须使用框架窗口的 SetIcon 函数)；

3. 不能在资源中设置对话框的标题，而需要通过框架窗口设置；

4. 你应该把对话框的风格设置如下：

可见的(WS_VISIBLE)

子窗口(WS_CHILD)

不能有最大化和最小化按钮、拖动条、系统菜单或其他标准的框架窗口风格。

在本书中，有许多例子程序中使用了对话框作为主窗口(比如列表框、编辑框和按钮控件的例子)。

4.8 界面对象与控件的联系

因为对话框是从资源文件中产生的，你不需要编写 C++程序来实现它和其上的控件对象的样子。虽然这种使用资源文件的方式使你能够可视化地创建对话框，但却使你的应用程序难以操作控件对象。ObjectWindows 体系克服了这个局限，它将对话框中的控件对象与界面对象联系起来，使你能够提供对消息的特殊响应(比如，你或许想让 edit 控件只允许输入一定范围的数字，或者你想让 button 控件在按下的时候改变状态)及利用数据成员和成员函数来操作控件对象，这样比用控件消息操作更加容易并且更适合面向对象的程序设计方式。

要将一个控件对象与控件元素联系起来，你需要定义一个指向控件对象的指针作为数据成员并在构造对话框对象时构造控件对象。控件类(比如 TButton 类)的构造函数需要一个指向父窗口对象的指针和控件的资源 ID 值。

与窗口对象不同，控件对象与界面元素相连只需要"构造"一步(窗口对象与界面元素相连需要"构造"和"创建"两步)，这是因为界面元素已经存在了，它是通过对话框资源装载的。你只需要告诉构造函数控件对象所对应的资源 ID 值就可以了。

创建对话框对象有一种省事的办法，就是让 ObjectWindows 为你做对话框对象创建的工作。通过在派生于 TWindow 的窗口类的构造函数里创建对象并将 this 指针指向父窗口，TWindow 的派生类建立一个子窗口表。如果把对话框对象当作父窗口的一个数据成员，你也可以完成同样的功能。然后，当 TWindow 的派生类创建时，它就创建子窗口表中所有的具有 wfAutoCreate 标志的子窗口并在父窗口显示的同时显示子窗口。

通过调用 EnableAutoCreate 函数可设置 wfAutoCreate 标志；调用 DisableAutoCreate 函数可取消 wfAutoCreate 标志。

TWindow 类调用 Create 函数来自动创建它的子窗口，这样所有的自动创建对话框都是无模式对话框。与其它的无模式对话框一样，如果使用自动创建的对话框，你必须把对话框设置为可见的，但是，你只能通过在资源文件中设置 WS_VISIBLE 标志，而不能使用 ShowWindow 函数来达到这个目的。

下面的程序代码显示了如何自动创建对话框：

```
class TMyFrame : public TFrameWindow
{
  public:
    TDialog *dialog;
    TMyFrame(TWindow *, const char far *);
};

TMyFrame::TMyFrame(TWindow *parent, const char far *title)
{
  Init(parent, true);
  dialog = new TDialog(this, "对话框");
  dialog->EnableAutoCreate();
}
```

4.9 管理对话框

与其他的子窗口不同，对话框在父窗口中会显示和删除多次，很少有与父窗口同时显示同时删除的。通常，一个应用程序对选择菜单项、鼠标按键、错误情况或其它事件进行响应时显示一个对话框，因此，你必须保证不在先前的对话框对象删除之前又创建一个新的对话框对象。记住当你在父窗口对象的构造函数里创建对话框对象或者把它当作父窗口的一个数据成员时，对话框对象就增加到子窗口表中，直到取消父窗口时才删除它。在对话框对象存在时你可以随时从对话框中接收数据。

几乎所有的对话框都象子窗口那样具有一系列控件，比如 edit 控件、list boxes 控件、buttons 控件以及其它控件，这些控件都是在对话框资源中定义的。

有两种方式可以把对话框对象和其上的控件联系起来。一种方式，对话框需要对控件进行操作，比如填充一个 list box 控件；另外一种方式，对话框需要处理和响应控件发出的消息，比如当用户从 list box 控件中选择了一项。Windows 定义了一系列控件消息，比如 list-box 控件消息包括 LB_GETTEXT、LB_GETCURSEL 和 LB_ADDSTRING。几乎没有必要通

过控件消息来与控件建立联系，因为 ObjectWindows 控件类提供了能够完成相同任务的成员函数。这里讨论建立与控件联系的目的是为了更好的理解这个过程。特定的控件消息指向特定控件，与 wParam 和 lParam 参数一起传输信息。对话框资源中的每一个控件都有一个 ID 值用来标记接收消息的控件。你可以使用 SendDlgItemMessage 发出控件消息，比如下面的成员函数用 LB_ADDSTRING 消息在 list box 控件中增加字符串：

```
void TTestDialog::FillListBox(const char far * string)
{
  SendDlgItemMessage(ID_LISTBOX, LB_ADDSTRING, 0, (LPARAM)string);
}
```

使用 TListBox::AddString 函数可取同样的工作，而且容易理解。

```
void listTDLGClient::SetupWindow ()
{
      TDialog::SetupWindow();

  ListBox1->AddString("面包会有的");
      ...
  ListBox2->AddString("面包会有的");
      ...
  ListBox3->AddString("面包会有的");
      ...
}
```

4.10 使用传输缓冲区

使用一个结构来描述对话框或窗口的控件状态，这个结构叫做传输缓冲区。比如你的应用程序可以弹出一个模式对话框，对话框关闭后从传输缓冲区中获得每一个控件的信息。然后，如果对话框又被启动了，你可以从传输缓冲区中传回控件状态。另外，你可以为每一个控件设置初始状态，你也可以在任何时刻设置或传输缓冲区数据，这样你就可以用先前的状态重新设置控件的状态。一个窗口或无模式对话框也可以用这种机制在任何时刻设置或恢复控件状态。

使用对话框的传输机制，下面是必要的三个步骤：

1. 定义传输缓冲区，为每一个你想传输的控件数据设置一个实例变量；
2. 定义与之相连的窗口或对话框；
3. 传输数据。

传输缓冲区是一个由每一个参与的控件组成的数据，其中的成员是实例变量。一个窗口或对话框也可以没有要传输的控件数据，比如，缺省时，按钮、组框和静态控件就不参与传输。控件类型决定了传输缓冲区中的数据类型，定义一个传输缓冲区需要给对话框和窗口中所有需要传输数据的控件定义一个实例变量，但并不需要给每一个控件都定义一个实例变量。比如下面的例子：

```
struct TransferStruct
{
   char              editCtlData[30];    //编辑框
   uint16            checkBoxData;       //检查按钮
   uint16            radioButtonData;    //radio 按钮
   TListBoxData      listBoxData;        //列表框
   TComboBoxData     comboBoxData;       //组合框
   TScrollBarData    scrollBarData;      //滚动条
};
```

每一种控件的传输缓冲区中存储着不同的信息，表 4-1 是每一种 ObjectWindows 控件传输缓冲区的描述：

表 4-1

控件类	传输类型	描　　述
Static	char array	以允许文本的最大长度定义字符数组，以 NULL 作为结束。缺省时，静态控件是不传输数据的，但可以显式开启此功能
Edit	char array	以允许文本的最大长度定义字符数组，以 NULL 作为结束
List box	TListBoxData	一个 TListBoxData 类的实例。TListBoxData 包含一系列数据成员，保存列表框的字符串、项目数据和选择索引
Combo box	TComboBoxData	一个 TComboBoxData 类的实例。TComboBoxData 包含一系列数据成员，保存列表框字符串、项目数据、选择索引和编辑区域的内容
Check box	uint16	包括检查框的状态：BF_CHECKED、BF_UNCHECKED 或 BF_GRAYED
radio 按钮	uint16	包括 radio 按钮的状态：BF_CHECKED、BF_UNCHECKED 或 BF_GRAYED
Scroll bar	TScrollBarData	一个 TScrollBarData 类的实例。包括三个数据成员：LowValue ：范围的最小值；HighValue：范围的最小值；Position：当前的指示位置

一个窗口或对话框如果使用这种传输机制，就必须以一定的顺序定义其中的控件对象，这个顺序与传输缓冲区中相关的数据成员的顺序完全相同。通过调用 SetTransferBuffer 函数来开启窗口或对话框的传输功能，这个函数传输一个传输缓冲区的指针。

多数情况下，从窗口中传出或向窗口中传入数据的过程是自动进行的，但你也可以在任何时刻显式地进行数据传输。

为了说明对话框的使用方法，我们编写了三个使用对话框的例子。第一个例子是以模式对话框方式运行；第二个例子是以无模式对话框方式运行。这两个例子说明了如何创建模式对话框和无模式对话框，我们在对话框中应用各种控件对象并对它们进行控制。例子中的另外一个重点是如何进行两种对话框的数据传输。第三个例子是一个对话框数据传输

的例子，它的数据传输方式与前面两个例子不同。在这三个例子中，定义的对话框都是相同的，见图 4-4。

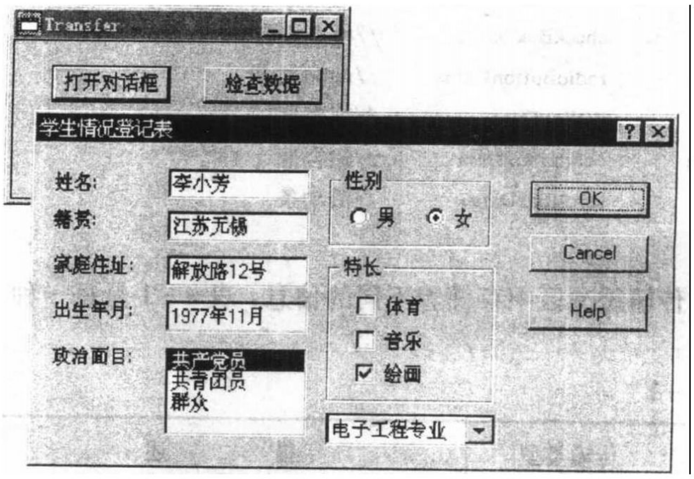


图 4-4　例子中的对话框

4.11　模式对话框程序实例

第一个例子演示模式对话框的数据传输。首先，使用 AppExpert 创建一个基于对话框的应用程序框架；使用 Workshop 修改对话框的外观，增加两个按钮；使用 WorkShop 创建两个新的对话框，测试数据传输；使用 ClassExpert 在项目文件中增加新的对话框类 TTransfer-Dialog(见图 4-5)；使用 ClassExpert 为控件增加变量名。

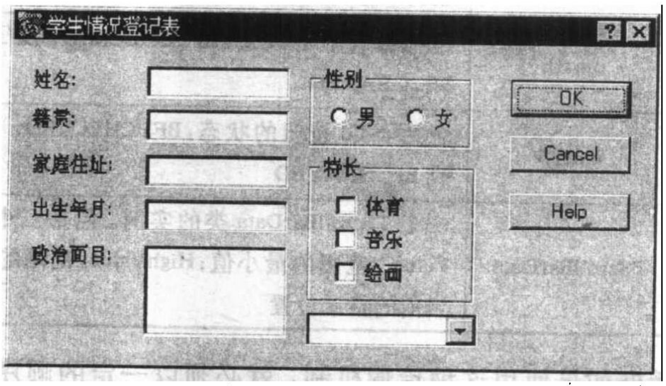


图 4-5　TTransferDialog 对话框

```
TTransferDialog::TTransferDialog(TWindow * parent, TResId resId, TModule * module)
:     TDialog(parent, resId, module)
{
//{{TTransferDialogXFER_USE}}
    tiyu = new TCheckBox(this, IDC_CHECKBOX1, 0);
    yinyue = new TCheckBox(this, IDC_CHECKBOX2, 0);
    huihua = new TCheckBox(this, IDC_CHECKBOX3, 0);
```

```
        zhuanye = new TComboBox(this, IDC_COMBOBOX1, 1);
        xingming = new TEdit(this, IDC_EDIT1, 255);
        jiguan = new TEdit(this, IDC_EDIT2, 255);
        jtzz = new TEdit(this, IDC_EDIT3, 255);
        csny = new TEdit(this, IDC_EDIT4, 255);
        zzmm = new TListBox(this, IDC_LISTBOX1);
        nan = new TRadioButton(this, IDC_RADIOBUTTON1, 0);
        nu = new TRadioButton(this, IDC_RADIOBUTTON2, 0);

    SetTransferBuffer(&TTransferDialogData);
  //{{TTransferDialogXFER_USE_END}}
  }
```

根据对话框要传输的控件数据定义了一个传输缓冲区的类型，其中包括：两个 radio 按钮的选择情况、几个编辑框的字符串缓冲区、三个检查框的选择情况、一个列表框数据、一个组合框数据等：

```
struct TTransferDialogXfer {
//{{TTransferDialogXFER_DATA}}
      TTransferDialogXfer();
      uint     tiyu;
      uint     yinyue;
      uint     huihua;
      TComboBoxData zhuanye;
      char     xingming[ 255 ];
      char     jiguan[ 255 ];
      char     jtzz[ 255 ];
      char     csny[ 255 ];
      TListBoxData zzmm;
      uint     nan;
      uint     nu;
//{{TTransferDialogXFER_DATA_END}}
};
```

在主窗口类中包含了对话框和它的传输缓冲区：

```
class TTransferDlgClient : public TDialog {
  public:
    TTransferDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TTransferDlgClient();
    TTransferDialog * aDialog;
    TTransferDialogXfer TTransferDialogData;
//{{TTransferDlgClientRSP_TBL_BEGIN}}
  protected:
  void OnButton1();
```

```
  void OnButton2();
//{{TTransferDlgClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TTransferDlgClient);
};    //{{TTransferDlgClient}}
```

其中，定义的两个函数 OnButton1 和 OnButton2 执行了两个对话框。第一个函数执行对话框得到的数据被第二个对话框显示出来。

缓冲区类型的构造函数中设置了对话框中的控件的初始状态：

```
TTransferDialogXfer::TTransferDialogXfer()
{
  nan = nu = FALSE;
  xingming[0] = 0;
  jiguan[0] = 0;
  jtzz[0] = 0;
  csny[0] = 0;
  tiyu = yinyue = huihua = 0;
  zzmm.AddString("共青团员");
  zzmm.AddString("共产党员");
  zzmm.AddString("群众");
  zzmm.Select(1);
  zhuanye.AddString("微电子专业");
  zhuanye.AddString("电子工程专业");
  zhuanye.AddString("计算机专业");
  zhuanye.AddString("科技英语专业");
  zhuanye.AddString("通信专业");
  zhuanye.Select(2);
}
```

在对话框类的构造函数中创建了全部的控件对象，它们的顺序与对话框缓冲区结构中的控件缓冲区顺序完全一致，是巧合吗？非也！你必须保证两者的顺序完全一致，这样才能进行正常的数据传输。另外，那些不需要操作的控件对象在 new 的左边不需要给什么控件对象指针赋值。

在对话框类中定义了需要操作的控件对象：

```
class TTransferDialog : public TDialog {
  public:
    TTransferDialog(TWindow * parent, TResId resId = DIALOG_1, TModule * module = 0);
    virtual ~TTransferDialog();

//{{TTransferDialogXFER_DEF}}
  protected:
    TCheckBox * tiyu;
    TCheckBox * yinyue;
    TCheckBox * huihua;
```

```
    TComboBox * zhuanye;
    TEdit * xingming;
    TEdit * jiguan;
    TEdit * jtzz;
    TEdit * csny;
    TListBox * zzmm;
    TRadioButton * nan;
    TRadioButton * nu;

//{{TTransferDialogXFER_DEF_END}}
};    //{{TTransferDialog}}
```

下面是模式对话框数据传输应用程序的部分代码，其项目文件见图 4-6。

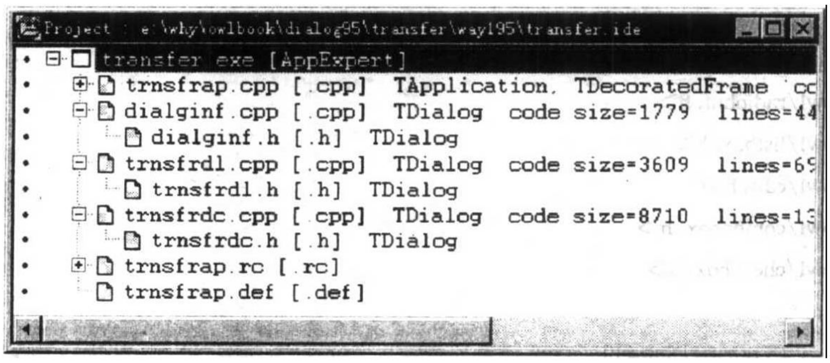


图 4-6 模式对话框应用程序项目文件

```
//------------------------------------------------------------
// FILE:          TrnsfrDC.h
// Class definition for TTransferDlgClient (TDialog).
//------------------------------------------------------------
#if !defined(trnsfrdc_h)            // Sentry, use file only if it's not already included.
#define trnsfrdc_h
#include "TrnsfrAp.rh"                  // Definition of all resources.
#include "trnsfrdl.h"

//{{TDialog = TTransferDlgClient}}
class TTransferDlgClient : public TDialog {
  public:    TTransferDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module =
0);
    virtual ~TTransferDlgClient();
    TTransferDialog * aDialog;
    TTransferDialogXfer TTransferDialogData;

//{{TTransferDlgClientRSP_TBL_BEGIN}}
  protected:
  void OnButton1();
```

```
    void OnButton2();
//{{TTransferDlgClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TTransferDlgClient);
};      //{{TTransferDlgClient}}

#endif      // trnsfrdc_h sentry.

//------------------------------------------------
//  FILE:           trnsfrdl.h
//  Class definition for TTransferDialog (TDialog).
//------------------------------------------------
#if !defined(trnsfrdl_h)            // Sentry, use file only if it's not already included.
#define trnsfrdl_h
#include <owl/dialog.h>
#include "TrnsfrAp.rh"                      // Definition of all resources.
#include <owl/radiobut.h>
#include <owl/listbox.h>
#include <owl/edit.h>
#include <owl/combobox.h>
#include <owl/checkbox.h>

//{{TDialog = TTransferDialog}}
struct TTransferDialogXfer {
//{{TTransferDialogXFER_DATA}}
    TTransferDialogXfer();
    uint      tiyu;
    uint      yinyue;
    uint      huihua;
    TComboBoxData zhuanye;
    char      xingming[ 255 ];
    char      jiguan[ 255 ];
    char      jtzz[ 255 ];
    char      csny[ 255 ];
    TListBoxData zzmm;
    uint      nan;
    uint      nu;
//{{TTransferDialogXFER_DATA_END}}
};

class TTransferDialog : public TDialog {
  public:
    TTransferDialog(TWindow * parent, TTransferDialogXfer& ts, TResId resId = DIALOG_1, TModule
* module = 0);
```

```
    virtual ~TTransferDialog();

  //{{TTransferDialogXFER_DEF}}
   protected:    TCheckBox * tiyu;
    TCheckBox * yinyue;
    TCheckBox * huihua;
    TComboBox * zhuanye;
    TEdit * xingming;
    TEdit * jiguan;
    TEdit * jtzz;
    TEdit * csny;
    TListBox * zzmm;
    TRadioButton * nan;
    TRadioButton * nu;

  //{{TTransferDialogXFER_DEF_END}}
  };     //{{TTransferDialog}}

  #endif      // trnsfrdl_h sentry.

  //-----------------------------------------------------------------
  //  FILE:           dialginf.h
  //  Class definition for TDialogInfo (TDialog).
  //-----------------------------------------------------------------
  #if ! defined(dialginf_h)           // Sentry, use file only if it's not already included.
  #define dialginf_h
  #include <owl/dialog.h>
  #include <cstring.h>
  #include "TrnsfrAp.rh"                   // Definition of all resources.
  #include <owl/edit.h>

  //{{TDialog = TDialogInfo}}

  class TDialogInfo : public TDialog {
    public:
      TDialogInfo(TWindow * parent, string& info, TResId resId = DIALOG_2, TModule * module = 0);
      virtual ~TDialogInfo();

  //{{TDialogInfoXFER_DEF}}
    protected:
      TEdit * DisplayInfo;
      string& Info;
  //{{TDialogInfoXFER_DEF_END}}
```

```
//{{TDialogInfoVIRTUAL_BEGIN}}
  public:
    virtual void SetupWindow();
//{{TDialogInfoVIRTUAL_END}}
};      //{{TDialogInfo}}

#endif      // dialginf_h sentry.

//----------------------------------------------------------------------
//  FILE:           TrnsfrDC.cpp
//  Source file for implementation of TTransferDlgClient (TDialog).
//----------------------------------------------------------------------
#include <owl/pch.h>
#include "dialginf.h"
#include "trnsfrdl.h"
#include "TrnsfrAp.h"
#include "TrnsfrDC.h"

//
// Build a response table for all messages/commands handled by the application.
//
DEFINE_RESPONSE_TABLE1(TTransferDlgClient, TDialog)
//{{TTransferDlgClientRSP_TBL_BEGIN}}
  EV_BN_CLICKED(IDC_BUTTON1, OnButton1),
  EV_BN_CLICKED(IDC_BUTTON2, OnButton2),
//{{TTransferDlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{TTransferDlgClient Implementation}}

//------------------------------------------------------------------
// TTransferDlgClient
// ~~~~~~~~~
// Construction/Destruction handling.
//
TTransferDlgClient::TTransferDlgClient(TWindow* parent, TResId resId, TModule* module)
: TDialog(parent, resId, module)
{
  aDialog = new TTransferDialog(this,TTransferDialogData);
}

TTransferDlgClient::~TTransferDlgClient()
```

```
{
  delete aDialog;
  Destroy();
}

void TTransferDlgClient::OnButton1()
{
  aDialog->Execute();
}

void TTransferDlgClient::OnButton2()
{

  string          displayInfo;
  TStringArray&   lbStrArray = TTransferDialogData.zzmm.GetStrings();
  TStringArray&   cbStrArray = TTransferDialogData.zhuanye.GetStrings();
  unsigned        lbNItems = lbStrArray.GetItemsInContainer();
  unsigned        cbNItems = cbStrArray.GetItemsInContainer();

  // mailing info.
  //
  displayInfo += "姓名:    ";
  displayInfo += TTransferDialogData.xingming;
  displayInfo += " \r\n";
  displayInfo += "性别:    ";  if (TTransferDialogData.nan)
      displayInfo += "男    ";
  else if (TTransferDialogData.nu)
      displayInfo += "女    ";
  else
      displayInfo += "性别不详    ";
  displayInfo += "  \r\n";
  displayInfo += "籍贯:    ";
  displayInfo += TTransferDialogData.jiguan;
  displayInfo += "  \r\n";
  displayInfo += "家庭地址:    ";
  if (strcmp(TTransferDialogData.jtzz, "") != 0)
  {
      displayInfo += TTransferDialogData.jtzz;
  }
  displayInfo += "  \r\n";
  displayInfo += "出生年月:    ";
  displayInfo += TTransferDialogData.csny;
  displayInfo += "  \r\n";
```

```
    // check boxes.
    //
    displayInfo += "特长包括：    ";
    displayInfo += "\r\n";
    if (TTransferDialogData.tiyu)
        displayInfo += "体育    \r\n";
    if (TTransferDialogData.yinyue)
        displayInfo += "音乐    \r\n";
    if (TTransferDialogData.huihua)
        displayInfo += "绘画    \r\n";
    displayInfo += "  \r\n";

    // list box strings.
    //
    if (lbNItems ! = 0)
    {
        displayInfo += "政治面目：    ";
        string sel;
        TTransferDialogData.zzmm.GetSelString(sel);
        displayInfo += sel;
        displayInfo += "  \r\n";
    }

    // combo box strings.
    //
    if (cbNItems ! = 0)
    {
        displayInfo += "所学专业：    ";
        displayInfo += TTransferDialogData.zhuanye.GetSelection();
    }

    TDialogInfo infoDialog(this, displayInfo);
    infoDialog.Execute();
}

//-------------------------------------------------------------------
//  FILE:          trnsfrdl.cpp
//  Source file for implementation of TTransferDialog (TDialog).
//-------------------------------------------------------------------
#include <owl/pch.h>
#include "trnsfrdl.h"
```

```
//{{TTransferDialog Implementation}}

TTransferDialogXfer::TTransferDialogXfer()
{
  nan = nu = FALSE;
  xingming[0] = 0;
  jiguan[0] = 0;
  jtzz[0] = 0;
  csny[0] = 0;
  tiyu = yinyue = huihua = 0;
  zzmm.AddString("共青团员");
  zzmm.AddString("共产党员");
  zzmm.AddString("群众");
  zzmm.Select(1);
  zhuanye.AddString("微电子专业");
  zhuanye.AddString("电子工程专业");
  zhuanye.AddString("计算机专业");
  zhuanye.AddString("科技英语专业");
  zhuanye.AddString("通信专业");
  zhuanye.Select(2);
}

TTransferDialog::TTransferDialog(TWindow * parent, TTransferDialogXfer& ts, TResId resId, TModule * module)
  :   TDialog(parent, resId, module)
{
//{{TTransferDialogXFER_USE}}
    tiyu = new TCheckBox(this, IDC_CHECKBOX1, 0);
    yinyue = new TCheckBox(this, IDC_CHECKBOX2, 0);
    huihua = new TCheckBox(this, IDC_CHECKBOX3, 0);
    zhuanye = new TComboBox(this, IDC_COMBOBOX1, 1);
    xingming = new TEdit(this, IDC_EDIT1, 255);
    jiguan = new TEdit(this, IDC_EDIT2, 255);
    jtzz = new TEdit(this, IDC_EDIT3, 255);
    csny = new TEdit(this, IDC_EDIT4, 255);
    zzmm = new TListBox(this, IDC_LISTBOX1);
    nan = new TRadioButton(this, IDC_RADIOBUTTON1, 0);
    nu = new TRadioButton(this, IDC_RADIOBUTTON2, 0);
  SetTransferBuffer(&ts);
//{{TTransferDialogXFER_USE_END}}
}

TTransferDialog::~TTransferDialog()
```

```
{
  Destroy();
}

//----------------------------------------------------------
//  FILE:            dialginf.cpp
//  Source file for implementation of TDialogInfo (TDialog).
//----------------------------------------------------------
#include <owl/pch.h>
#include "dialginf.h"

//{{TDialogInfo Implementation}}

TDialogInfo::TDialogInfo(TWindow * parent, string& info, TResId resId, TModule * module)
:    TDialog(parent, resId, module), Info(info)
{
//{{TDialogInfoXFER_USE}}
    DisplayInfo = new TEdit(this, IDC_EDIT6, 255);
//{{TDialogInfoXFER_USE_END}}
}

TDialogInfo::~TDialogInfo()
{
  Destroy();
}

void TDialogInfo::SetupWindow()
{
  TDialog::SetupWindow();
  DisplayInfo->Insert(Info.c_str());
}
```

4.12　无模式对话框程序实例

无模式对话框与模式对话框所不同的地方在于：在主窗口类的事件响应表中包含有对话框关闭的事件(因为在无模式对话框关闭以前主窗口也接收消息)。

在主窗口中增加了两个按钮的对象变量，使用它们可以操作按钮的属性：

```
TTransferDlgClient::TTransferDlgClient(TWindow * parent, TResId resId, TModule * module)
:    TDialog(parent, resId, module)
{
//{{TTransferDlgClientXFER_USE}}
```

```
    Button1 = new TButton(this, IDC_BUTTON1);
    Button2 = new TButton(this, IDC_BUTTON2);
  //{{TTransferDlgClientXFER_USE_END}}

   aDialog = new TTransferDialog(this,TTransferDialogData);
  }
```

当对话框打开时，为了避免用户重复按下“打开对话框”按钮或按下“检查数据”按钮，程序将这两个按钮设置为 Disable，并且使用 Create 函数创建对话框界面元素。

```
void TTransferDlgClient::OnButton1()
{
  Button1->EnableWindow(false);
  Button2->EnableWindow(false);
  aDialog->Create();
}
```

当对话框关闭时，我们希望数据被传输，并且两个按钮恢复使用，当主窗口并不能知道对话框关闭这样的事情，这就需要定义新的事件和响应函数。

const WM_DIALOG_CLOSED = WM_USER + 100;

当对话框关闭时就向父窗口发出消息：

```
void TTransferDialog::EvDestroy()
{
  // tell application that dialog is closing.
  Parent->PostMessage(WM_DIALOG_CLOSED);
  TDialog::EvDestroy();
}
```

主窗口事件响应表中定义了对话框关闭事件的响应。当对话框关闭时，传输对话框事件，恢复两个按钮的状态。

```
void TTransferDialog::CloseWindow(int retVal)
{
  TransferData(tdGetData);
  TDialog::CloseWindow(retVal);
}
```

无模式对话框并不常用，而且主要的方式与模式对话框相似，这里不再给出代码。

4.13 传输缓冲区程序实例

第三个例子中的对话框缓冲区比较特殊，它是用静态数据成员来实现对话框的数据传输。我们知道，static 关键字在 C 和 C++中具有很多用途。

在 C 语言中，static 包括两种含义：

1. 在函数中使用 static 可以使数据被静态的分配而不是放在堆栈中。

2. 用 static 说明外部的数据和过程具有局部的连接性，不能被当前文件以外的过程直接访问。

在 C++中，它保留了 C 语言的这两种含义，另外还增加了两种含义：

1. 在类中说明为静态数据成员，那么类的这个数据成员在程序的生命过程中都将存在，可以随时访问，不管这个类有没有实例对象。

2. 在一个程序中，类的静态成员只有一份拷贝，不管存在多少对象，它们都是同样的值。

在前面的两个例子中，我们总是将对话框的数据传输缓冲区定义在主窗口中，在构造对话框对象时通过对话框的构造函数的参数将缓冲区的地址告诉对话框。为什么要将缓冲区定义在主窗口中？因为主窗口的寿命比对话框长（从程序开始到程序结束），这样缓冲区中的信息就不会丢失，而且具有连续性（你会发现对话框关闭后再打开，上次输入的数据依然保留）。为什么不把对话框的数据传输缓冲区作为对话框类的数据成员呢？通常，如果将缓冲区定义在对话框类中，当对话框关闭时，数据缓冲区中的数据就不能被使用了。

现在我们有办法将对话框数据传输缓冲区作为对话框类的数据成员了：我们将它作为对话框类中的静态数据成员。这样，即使对话框对象删除了我们依然可以访问缓冲区的内容，而且下一个对话框打开时会保存前一个对话框的输入内容。

```
class TTransferDialog : public TDialog {
  public:
          ...
      static TTransferStruct TransferStruct;
private:
      TListBox *          ListBox;
      TComboBox *         ComboBox;
      TEdit *             ListBoxData;
      TEdit *             ComboBoxData;
      TScrollBar *        ScrollBar;
};
```

在对话框的构造函数中设置了传输缓冲区：

```
TTransferDialog::TTransferDialog(TWindow * parent, int resId)
  : TDialog(parent, resId)
{
          ...
  SetTransferBuffer(&TransferStruct);
}
```

你可能发现在构造函数中并没有初始化缓冲区数据成员，为什么？因为必须显式地初始化静态数据成员。下面的语句初始化了对话框中的缓冲区：

TTransferStruct TTransferDialog::TransferStruct;

当程序的任何地方需要访问对话框的缓冲区时，我们可以通过 TTransferDialog::TransferStruct 来实现访问，它并不依靠某一个对话框的对象。此例的程序代程与前面例子中的

代码相似，故略之。

4.14 输入框

OWL 提供一个用于用户输入的对话框：输入框(见图 4-7)。使用输入框时你需要在资源文件中建立一个输入框的资源：

```
IDD_INPUTDIALOG DIALOG 20, 24, 180, 64
STYLE WS_POPUP | WS_CAPTION | DS_SETFONT
FONT 8, "Helv"
{
  LTEXT "", ID_PROMPT, 10, 8, 160, 10, SS_NOPREFIX
     CONTROL "", ID_INPUT, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP | ES_AUTOHSCROLL, 10, 20, 160, 12
  DEFPUSHBUTTON "&OK", IDOK, 47, 42, 40, 14
  PUSHBUTTON "&Cancel", IDCANCEL, 93, 42, 40, 14
}
```

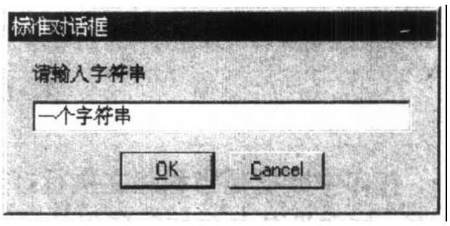


图 4-7 标准的输入框

如果使用 AppExpert 来创建你的应用程序，它将自动地在资源文件中创建一个输入框资源。

输入框类包含了以下这些数据成员：

buffer：指向传输用户输入文本的缓冲区；

BufferSize：用户的输入文本缓冲区的大小；

prompt：指向输入对话框的说明文字。

输入对话框创建时，同时创建了它的标题条、说明文字和缺省的编辑框。

输入框的构造函数是：

```
TInputDialog(TWindow * parent,
             const char far  * title,
             const char far  * prompt,
             char far * buffer,
             int buffersize,
             TModule * module = 0,
             TValidator * valid = 0)
```

其中，parent 是输入框的父窗口指针；title 是输入框的窗口标题；prompt 是输入框的说明文字；buffer 是输入文本缓冲区；buffersize 是缓冲区的大小；module 是模块指针；而 valid 是 TValidator 类的指针，TValidator 类的用法参见有关编辑框控件的章节。

下面是使用输入框的例子(不是完整程序)：

```
......
char inputText[40]="一个字符串";
```

```
if ((TInputDialog(this, "标准输入框",
                  "请输入一个字符串:",
                  inputText,
                  sizeof(inputText))).Execute() == IDOK)
{
    ……  //在这里处理输入的字符串
}
……
```

4.15 打印中断框

打印中断框是当程序在执行打印任务时经常使用的对话框，它可以显示正在打印的文件名、打印设备名和打印所用的接口(见图 4-8)。

TPrinterAbortDlg 类封装了打印中断框的特性。TPrinterAbortDlg 包含了三个静态文本控件，标题的 ID 值为 101，打印设备的 ID 值为 102，输出端口的 ID 值为 103。在字符串中必须加上"%s"，以便被标题、打印设备和输出端口的字符串取代。

图 4-8　标准的打印中断框

如果程序需要使用打印中断框，你就需要在资源文件中创建这个对话框的资源：

```
IDD_ABORTDIALOG DIALOG 84, 51, 130, 60
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "正在打印"
{
  CTEXT "Now printing", -1, 0, 8, 130, 8
  CTEXT "目前打印的是第%d 页", ID_PAGE, 0, 8, 130, 8, SS_CENTER | NOT WS_VISIBLE | WS_
GROUP
  CTEXT "文件:'%s'", ID_TITLE, 0, 16, 130, 8
  CTEXT "", ID_PORT, 0, 24, 130, 8, SS_CENTER | NOT WS_VISIBLE | WS_GROUP
  CTEXT "%s 打印机在端口%s", ID_DEVICE, 0, 24, 130, 8
  CONTROL "Button", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_
TABSTOP, 48, 36, 33, 21
}
```

通常，不需要显式地创建打印中断框，你可以使用 TPrinter 和 TPrintout 来使用这个对话框。

4.16 通用对话框

为了方便地使用 Windows 提供的通用对话框，OWL 从 TDialog 类中派生出来一个通用对话框类 TCommonDialog，它是各种通用对话框类的虚基类。ObjectWindows 中从这个类派

生出来的类有：

TChooseColorDialog	让用户选择颜色的模式对话框；
TChooseFontDialog	让用户选择字体的模式对话框；
TReplaceDialog	让用户取代文字的无模式对话框；
TFindDialog	让用户寻找文字的无模式对话框；
TFileOpenDialog	让用户打开特定文件的模式对话框；
TFileSaveDialog	让用户以特定文件存盘的模式对话框；
TPrintDialog	让用户进行打印设置的模式对话框。

每一个通用对话框类都有一个伴随的数据类 TData，它存储着对话框特性参数和用户的使用数据。比如，TChooseColorDialog::TData 类保存了用户选择的颜色参数。事实上，在构造一个通用对话框对象之前就构造了一个 TData 对象，填充 TData 对象的数据成员。然后执行一个模式对话框或创建一个无模式对话框。

当你的程序使用这些通用对话框时，你不需要在资源文件中创建对话框的资源。

4.17 颜色选择对话框

图 4-9 所示为颜色选择对话框，颜色选择对话框(TChooseColorDialog 类)是一个模式对话框，它允许用户选择调配颜色。TChooseColorDialog 类可以无模式显示，把对话框的父窗口作为一个不可见的弹出窗口，将这个弹出窗口作为应用程序主窗口的子窗口。

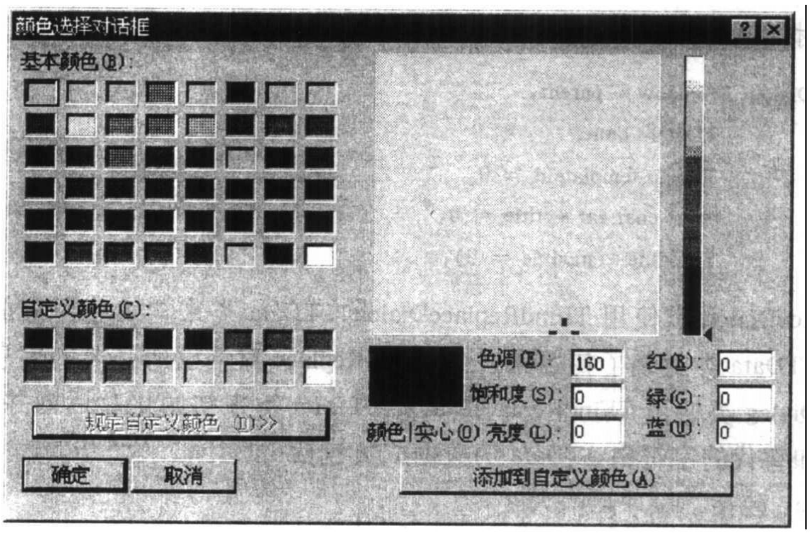


图 4-9 标准的颜色选择对话框

颜色选择对话框的构造函数是：

```
TChooseColorDialog(TWindow * parent,
        TData& data,
        TResId templateID = 0,
        const char far * title = 0,
        TModule * module = 0);
```

TChooseColorDialog 类使用 TChooseColorDialog::TData 类来初始化用户的颜色选择。TData 结构定义了初始化颜色选择对话框的必要信息。它的数据成员包括：

TColor Color：设置对话框创建时颜色选择的初始值；

TColor * CustColors：指向一个包含 16 种颜色的数组，它们显式在对话框中。

uint32 Error：保存错误信息。

uint32 Flags：标志控制着对话框的型式和功能。

Flags 的取值和意义如表 4-2 所示。

表 4-2

取值	意　　义
CC_FULLOPEN	当对话框创建时显示全部的对话框
CC_PREVENTFULLOPEN	关闭"Define Custom Colors" 按钮
CC_RGBINIT	使对话框使用 rgbResult 中定义的颜色作为初始颜色选择
CC_SHOWHELP	使对话框显示 Help 按钮

4.18 查找和替换对话框

TFindReplaceDialog 类是 TFindDialog 类(查找文本对话框)和 TReplaceDialog(替换文本对话框)两个对话框的父类。

查找替换对话框的构造函数是：

```
TFindReplaceDialog(TWindow * parent,
                   TData& data,
                   TResId templateId = 0,
                   const char far * title = 0,
                   TModule * module = 0);
```

TFindReplaceDialog 类使用 TFindReplaceDialog::TData 类来初始化用户的设置。TFindReplaceDialog::TData 类封装了初始化一个 TFindReplace 对话框所必要的信息。TFindDialog 类和 TReplaceDialog 类使用 TFindReplaceDialog::TData 类来初始化对话框并接收用户的输入，比如查找和替代的字符串。TData 的数据成员包括：

int BuffSize：包括了缓冲区的大小；

uint32 Error：保存了错误信息；

char * FindWhat：包含查找的字符串；

char * ReplaceWith：包含要替换的字符串；

uint32 Flags：Flags 描述对话框中控件按钮的状态和动作。

其中，Flag 可以是表 4-3 所列值的组合。

表 4-3

意 义	取 值
FR_DOWN	在查找对话框的方向组中选择 Down 按钮
FR_HIDEMATCHCASE	Match Case 检查框被隐藏
FR_HIDEWHOLEWORD	Whole Word 检查框被隐藏
FR_HIDEUPDOWN.	Up 和 Down 按钮被隐藏
FR_MATCHCASE	Match Case 检查框为选择状态
FR_NOMATCHCASE	Match Case 检查框为非选择状态
FR_NOUPDOWN	Up 和 Down 按钮被禁止使用
FR_NOWHOLEWORD	Whole Word 检查框被禁止使用
FR_REPLACE	Replace 按钮在替代对话框中使用
FR_REPLACEALL	Replace All 按钮在替代对话框中使用
FR_WHOLEWORD	Whole Word 检查框的状态为选择状态

查找对话框

查找对话框 TFindDialog 类创建一个无模式对话框，让用户查找文本(见图 4-10)。TFindDialog 提供一个注册的消息与父窗口联系。因为是从 TFindReplaceDialog 类派生出来的，TFindDialog 使用结构 TFindReplaceDialog::TData 根据用户输入的值(比如，要查找的文本)来初始化对话框。

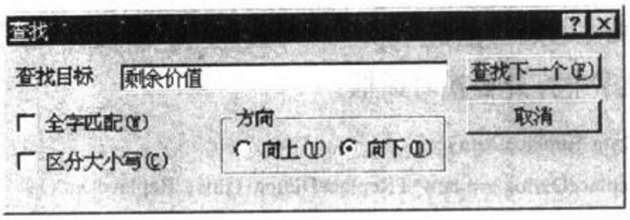


图 4-10 具有 3D 风格的查找对话框

查找对话框的构造函数是：

```
TFindDialog(TWindow * parent,
        TData& data,
        TResId templateId = 0,
        const char far * title = 0,
        TModule * module = 0);
```

下面的代码在程序中打开查找对话框：

```
TFindDialog::TData FindData;
TFindDialog * FindDialog = new TFindDialog(this, FindData);
FindDialog->Create();
```

关于一个无模式对话框的具体用法可以参考有关无模式对话框的章节。查找对话框通常于文本编辑的情况，而 OWL 的文本编辑类里已经封装了使用查找对话框的功能，所以通常我们不需要在程序中显式地调用这个对话框。

替换对话框

替换对话框 TReplaceDialog 类创建一个无模式对话框，让用户替换文本(见图 4-11)。由于是无模式对话框，用户可以一边编辑文本一边进行文本替换。因为是从 TFindReplaceDialog 类派生出来的，所以 TReplaceDialog 使用结构 TFindReplaceDialog::TData 并根据用户输入的值(比如，要替换的文本)来初始化对话框。

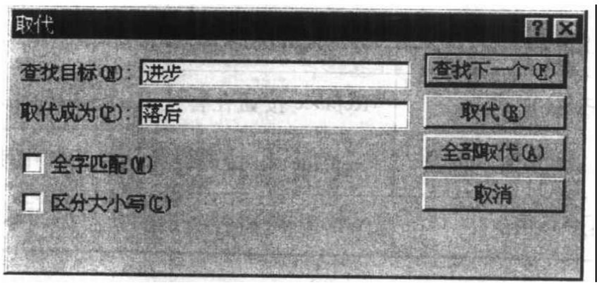


图 4-11　具有 3D 风格的替换对话框

替换对话框的构造函数是：

```
TReplaceDialog(TWindow * parent,
               TData& data,
               TResID templateName=0,
               const char far * title=0,
               TModule * module=0);
```

下面的代码在程序中打开替换对话框：

```
TReplaceDialog ::TData ReplaceData;
TReplaceDialog * ReplaceDialog = new TReplaceDialog (this, ReplaceData);
ReplaceDialog->Create();
```

关于一个无模式对话框的具体用法可以参考有关无模式对话框的章节。与查找对话框一样，替换对话框通常于文本编辑的情况，而 OWL 的文本编辑类里已经封装了使用替换对话框的功能，所以通常我们不需要在程序中显式地调用这个对话框。

4.19 字体选择对话框

字体选择对话框 TChooseFontDialog 是模式对话框，让用户选择字体、字体风格(比如加粗或倾斜)、字体大小、效果(比如删除线或下划线)和字体颜色(见图 4-12)。TChooseFontDialog 对象可作为无模式对话框显示，把对话框的父窗口作为一个不可见的弹出窗口，

将这个弹出窗口作为应用程序主窗口的子窗口。TChooseFontDialog 使用 TChooseFontDialog ::TData 结构来初始化对话框的用户选择字体。

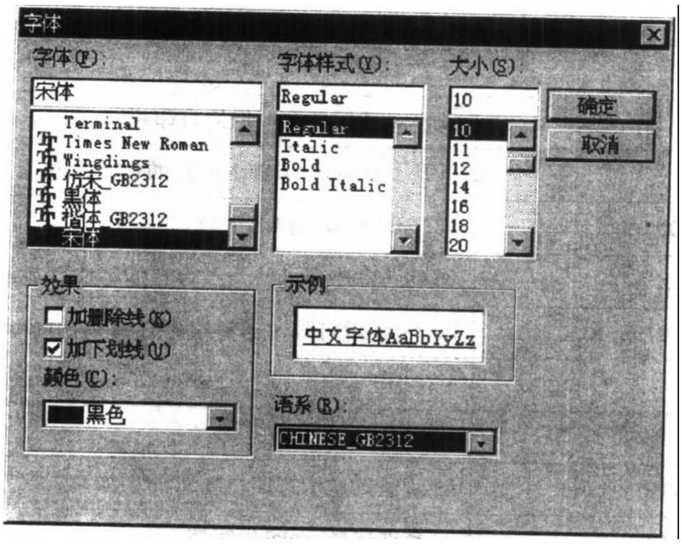


图 4-12 标准字体选择对话框

字体选择对话框的构造函数是：

```
TChooseFontDialog(TWindow * parent,
                  TData& data,
                  TResID templateID = 0,
                  const char far * title = 0,
                  TModule * module = 0);
```

TChooseFontDialog::TData 类定义了初始化字体选择对话框的必要信息。它包括的数据成员有：

TColor Color：设置对话框创建时字体颜色选择的初始值

HDC DC：字体获得的设备上下文句柄

uint32 Error：保存了错误信息

uint32 Flags：标志控制着对话框的型式和功能

uint16 FontType：字体类型或名字

LOGFONT LogFont：字体风格

int PointSize：字体大小

int SizeMax：字体允许的最大尺寸

int SizeMin：字体允许的最小尺寸

char far * Style：字体风格(比如，bold、italic、underline 或 strikeout)。

其中，Flages 的取值可以是表 4-4 中所列值的组合。

表 4-4

取　值	意　　义
CF_APPLY	打开并显示 Apply 按钮
CF_ANSIONLY	设置 ChooseFontDialog 结构只显示 ANSI 字符集的字体
CF_BOTH	设置对话框显示可用的打印机字体和屏幕字体
CF_EFFECTS	打开 strikeout、underline 和颜色效果功能
CF_FIXEDPITCHONLY	只打开 fixed—pitch 字体
CF_FORCEFONTEXIST	如果用户选择了一个不存在的字体就显示出错信息
CF_INITTOLOGFONTSTRUCT	LogFont 指向对话框初始化时的 LOGFONT 结构
CF_LIMITSIZE	将字体选择限制在 SizeMin 和 SizeMax 之间
CF_NOSIMULATIONS	不允许 GDI 字体仿真
CF_PRINTERFONTS	对话框只显示打印机要求的字体
CF_SCALABLEONLY	只允许选择 scalable 字体
CF_SCREENFONTS	只显示系统提供的屏幕字体
CF_TTONLY	只显示和允许选择 TrueType 字体
CF_USESTYLE	按照一个缓冲区中定义的字体风格设置字体
CF_WYSIWYG	只允许选择打印机和屏幕可用的字体
CC_SHOWHELP	使对话框显示 Help 按钮

4.20 打印对话框

打印对话框 TPrintDialog 显示一个模式打印或打印设置对话框(见图 4-13，图4-14)。打印对话框让用户设置一个打印任务的特性;打印设置对话框让用户配置打印机和设置扩展打印任务特性。TPrintDialog 使用 TPrintDialog::TData 结构来初始化用户的打印机设置对话框，比如，打印页数、输出设备等等。

打印对话框类构造函数是:

```
TPrintDialog(TWindow * parent,
        TData& data,
        const char far * printTemplateName=0,
        const char far * setupTemplateName=0,
        const char far * title=0,
        TModule * module=0);
```

TPrintDialog ::TData 对象包括了打印设置，比如，打印的份数、打印页数以及输出设备名等等。

通常，不必在程序中显式地调用打印对话框，可以使用 TPrinter 和 TPrintout 来提供打印机对话框。下面的例子说明如何通过 TPrinter 类来使用打印对话框和打印设置对话框:

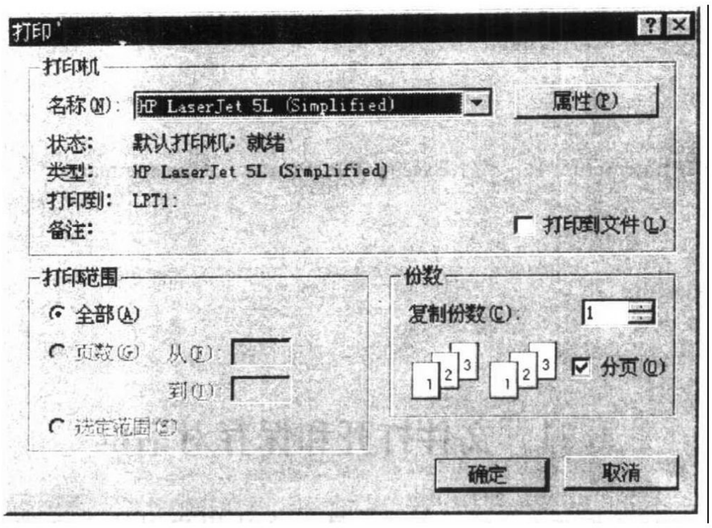


图 4-13 打印对话框

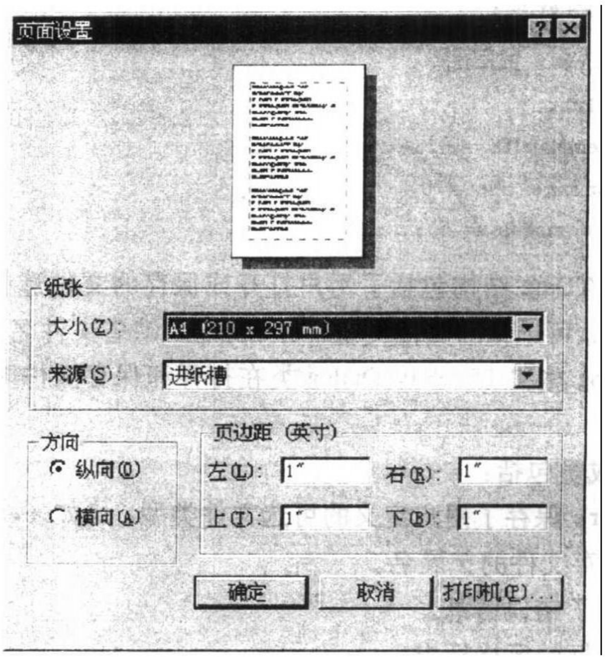


图 4-14 打印设置对话框

```
TPrinter * Printer;
Printer = new TPrinter;

void TRulerWin::CmFilePrint()
{
  if (Printer) {
      TWindowPrintout printout("ABC", this);
      printout.SetBanding(TRUE);
```

```
        Printer->Print(this, printout, TRUE);
    }
}

void
TRulerWin::CmFilePrinterSetup()      // Execute File:Printer-setup command
{
  if (Printer)
    Printer->Setup(this);
}
```

4.21　文件打开和保存对话框

文件打开和保存对话框 TOpenSaveDialog 是一个让用户以特定文件名打开和保存文件的对话框的基类。TOpenSaveDialog 包含了一个 结构 TData，将此结构传输给 TOpenSaveDialog 的构造函数，然后就以模式或无模式对话框的方式创建对话框。

文件对话框的构造函数是：

```
TOpenSaveDialog(TWindow * Parent,
                TData& data,
                TResID templateID = 0,
                const char far * title = 0,
                TModule * module = 0);
```

TOpenSaveDialog::TData 结构包括了用户打开或保存的文件选择信息。它保存了文件名、文件长度、扩展名、可选择的文件类型、初始路径、缺省文件名、扩展名以及全部的出错信息。TFileOpenDialog 类和 TFileSaveDialog 类在打开和保存文件时都使用保存在这个结构中的信息。

Data 结构的数据成员包括：

char * CustomFilter：保存了用户定义的可选文件类型，比如 *.txt 或 *.cpp。

char * DefExt：保存文件的扩展名。

uint32 Error：保存了错误信息。

uint32 Flags：对话框的风格标志。

char * FileName：保存打开或保存的文件名。

char * Filter：保存定义的可选文件类型。

int FilterIndex：可选文件类型的索引。

char * InitialDir：初始化时的路径。

其中，Flag 包括了表 4-5 中所列值的组合。

表 4-5

取　　值	意　　　　义
OFN_HIDEREADONLY	隐藏只读检查框
OFN_FILEMUSTEXIST	让用户只选择在文件列表框中已存在文件
OFN_PATHMUSTEXIST	让用户只能输入正确的路径，否则提示错误
OFN_NOVALIDATE	不检查文件名
OFN_NOCHANGEDIR	设置对话框创建时的当前路径
OFN_ALLOWMULTISELECT	在文件列表框中允许多个选择
OFN_CREATEPROMPT	当用户创建一个新文件时给出提问
OFN_EXTENSIONDIFFERENT	表示用户输入的文件名的扩展名与 DefExt 要求的不同
OFN_NOREADONLYRETURN	返回的文件没有只读的属性并且路径没有写保护
OFN_NOTESTFILECREATE	对话框关闭后创建文件
OFN_OVERWRITEPROMPT	Save As 对话框显示提问是否覆盖一个已存在的文件
OFN_SHAREAWARE	如果发生文件共享问题，返回所给的文件名
OFN_SHOWHELP	在对话框上显示 Help 按钮

文件打开对话框

文件打开对话框 TFileOpenDialog 是一个让用户打开特定文件的模式对话框。使用这个对话框响应当用户选择 File|Open 菜单项时发出的 CM_FILEOPEN 命令。TFileOpenDialog 类使用 TOpenSave::TData 结构来初始化文件打开对话框。

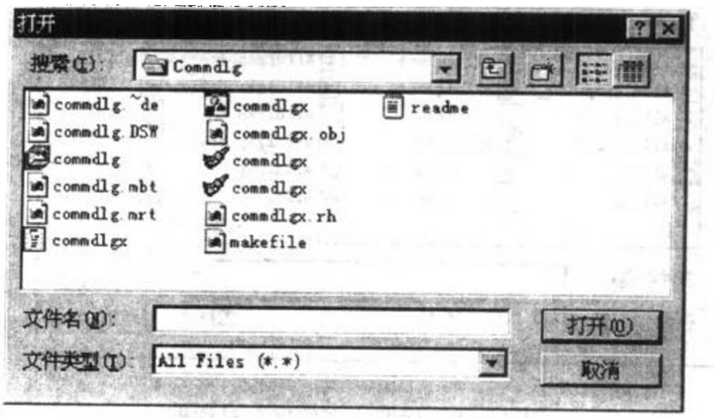


图 4-15　标准的文件打开对话框

文件打开对话框的构造函数是：

```
TFileOpenDialog(TWindow * parent,
                TData& data,
                TResID templateID = 0,
                const char far * title,
                TModule * module = 0);
```

下面的程序代码中使用了文件打开对话框：

```
TOpenSaveDialog::TData FileData;

FileData.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY
               | OFN_OVERWRITEPROMPT;
FileData.SetFilter("位图文件 (*.bmp)|*.bmp|");

#if defined(BI_PLAT_WIN32)
  if (TSystem::IsNT() || TSystem::IsWin95())
    FileData.Flags |= OFN_LONGNAMES;

  if (TSystem::IsWin95())
    FileData.Flags |= OFN_EXPLORER;
#endif

*FileData.FileName = 0;
if (TFileOpenDialog(this, FileData, 0, "打开文件").Execute() == IDOK)
      OpenFile();
```

文件保存对话框

文件保存对话框 TFileSaveDialog 是一个让用户以特定文件名存盘的模式对话框(见图 4-16)。使用这个对话框是响应 CM_FILESAVEAS 命令，当用户选择 File|Save 菜单项时发出这个命令。TFileSaveDialog 类使用 TOpenSave::TData 结构来初始化文件存盘对话框。

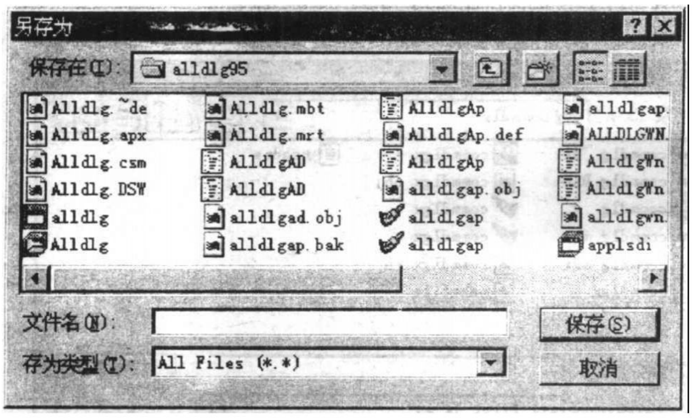


图 4-16　标准的文件保存对话框

文件保存对话框的构造函数是：

```
TFileSaveDialog(TWindow * parent,
                TData& data,
                TResID templateID = 0,
                const char far * title =0,
                TModule * module = 0);
```

下面的程序代码中使用了文件保存对话框：

```
TOpenSaveDialog::TData FileData;

FileData.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY
                | OFN_OVERWRITEPROMPT;
FileData.SetFilter("位图文件 (*.bmp)|*.bmp|");

*FileData.FileName = 0;
if (TFileSaveDialog(this, FileData, 0, "保存文件").Execute() == IDOK)
    SaveFile();
```

4.22 通用对话框程序实例

下面的例子充分地应用了通用对话框，其中包括：文件打开对话框、文件保存对话框、打印文件对话框、打印设置对话框、打印中断对话框、字符串输入对话框、字体选择对话框、颜色选择对话框等。

本例中，应用程序主窗口的客户窗口是一个从 TWindow 类派生出来的类 TcomdlgWindow，它的特有功能包括响应几乎全部的菜单项命令(除了“关于”是应用程序类响应的以外)，包括“新建文件”、“打开文件”、“保存文件”、“文件另存为”、“打印文件”、“打印设置”、“输入字符串”、“选择颜色”、“选择字体”等。

菜单项命令与响应函数的对应关系见表 4-6。

表 4-6

菜单项命令	响应函数	意　义
CM_BKCOLOR	TcomdlgWindow::CmColor	选择颜色
CM_FILENEW	TcomdlgWindow::CmNew	新建文件
CM_FILEOPEN	TcomdlgWindow::CmOpen	打开文件
CM_FILESAVE	TcomdlgWindow::CmSave	保存文件
CM_FILESAVEAS	TcomdlgWindow::CmSaveAs	文件另存为
CM_FONT	TcomdlgWindow::CmFont	选择字体
CM_PRINT	TcomdlgWindow::CmPrint	文件打印
CM_PRNSET	TcomdlgWindow::CmPntSet	打印设置
CM_STRING	TcomdlgWindow::CmString	输入字符串
CM_HELPABOUT	TcomdlgApp::CmHelpAbout	关于程序

客户窗口类 TcomdlgWindow 中包括了字体类 TFont 的指针 Font，用来保存所选的字体，TChooseFontDialog::TData FontData 结构用来保存字体选择对话框的信息；Printer 指针指向 TPrinter 类的对象，用来完成文件对应。bool 类型变量 IsDirty 和 IsNewFile 用来保存当前文件的状态，说明当前文件是否是一个新文件(没有文件名)；是否被改变了(字体、字符串、颜色的改变)。对文件操作通常需要监测这两个变量以确定是否应该存盘。字符串 Str

用来保存要显示的字符串。

```
class TcomdlgWindow : public TWindow
{
public:
    TcomdlgWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TcomdlgWindow ();
    TFont * Font;
    TChooseFontDialog::TData FontData;
    char Str[20];
    TOpenSaveDialog::TData FileData;
    TPrinter * Printer;
    void Paint (TDC& dc, bool erase, TRect& rect);
    virtual bool CanClose ();
protected:
    bool IsDirty, IsNewFile;
    void CmColor ();
    void CmNew ();
    void CmOpen ();
    void CmSave ();
    void CmSaveAs ();
    void CmFont ();
    void CmPrint ();
    void CmPntSet ();
    void CmString ();
    void NewFile ();
    void OpenFile ();
    void SaveFile ();
    void SaveAsFile ();
    void EvSize (uint sizeType, TSize& size);
DECLARE_RESPONSE_TABLE(TcomdlgWindow);
};
```

客户窗口类 TcomdlgWindow 的构造函数中进行了一系列初始化工作，设置窗口的背景颜色为白色；字体指针为 0；初始化的字符串 Str 为“通用对话框测试”；文件数据的格式设置；初始化两个布尔变量并构造了 TPrinter 类的对象。

```
TcomdlgWindow::TcomdlgWindow(TWindow * parent,const char far * title,TModule * module)
        : TWindow(parent, title, module)
{
    SetBkgndColor(RGB(255,255,255));
    Font = 0;
    sprintf(Str,"通用对话框测试");
    FileData.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
```

```
            OFN_OVERWRITEPROMPT;
        FileData.SetFilter("字符串格式 (*.why)|*.why|All Files (*.*)|*.*|");
        IsNewFile = true;
        IsDirty = false;
        Printer = new TPrinter;
}
```

CanClose 函数用来检验窗口关闭的条件是否满足。如果 IsDirty 变量为真(文件被改变了)就需要提示用户存盘。

```
bool TcomdlgWindow::CanClose ()
{
  if (IsDirty)
      switch(MessageBox("需要存盘吗?", "文件改变了",
                                    MB_YESNOCANCEL | MB_ICONQUESTION))
      {
          case IDCANCEL:
              return false;
          case IDYES:
              CmSave();
      }
  return true;
}
```

NewFile 函数用来新建一个文件，设置字符串为空。设置 IsNewFile 变量为真，IsDirty 变量为否。提供 Invalidate 函数重绘窗口。

```
void TcomdlgWindow::NewFile ()
{
        sprintf(Str,"");
        IsNewFile =true;
        IsDirty = false;
        SetBkgndColor(RGB(255,255,255));
        Invalidate();
}
```

SaveAsFile 函数将打开一个文件保存对话框让用户输入一个文件名，然后调用 SaveFile 函数保存文件。

```
void TcomdlgWindow::SaveAsFile ()
{
  if (IsNewFile)
      strcpy(FileData.FileName, "");

  if ((TFileSaveDialog(this, FileData, 0, "文件另存为")).Execute() == IDOK)
      SaveFile();
```

```
}
```

OpenFile 函数根据所给的文件名打开一个文件。然后重绘窗口。

```
void TcomdlgWindow::OpenFile ()
{
  ifstream is(FileData.FileName);

  if (! is)
      MessageBox("无法打开文件","文件错误", MB_OK | MB_ICONEXCLAMATION);
  else
          is >> Str;
  IsNewFile = IsDirty = false;
  Invalidate();
}
```

SaveFile 函数根据所给的文件名保存文件。

```
void TcomdlgWindow::SaveFile ()
{
  ofstream os(FileData.FileName);
  if (! os)
      MessageBox("无法打开文件","文件错误", MB_OK | MB_ICONEXCLAMATION);
  else
  {
      os << Str;
      IsNewFile = IsDirty = false;
  }
}
```

EvSize 函数响应窗口大小改变的消息并重绘窗口。

```
void TcomdlgWindow::EvSize (uint sizeType, TSize& size)
{
    TWindow::EvSize(sizeType, size);
    Invalidate();
}
```

CmColor 函数用来响应“选择颜色”菜单项命令，该函数定义了颜色选择对话框的数据结构 TChooseColorDialog::TData choose 用来保存对话框数据；一个颜色类型的数组用来保存对话框中的用户定义颜色；

```
static TColor custColors[16] =
{
    ...
};
```

初始化的当前选择的颜色为黑色的程序代码为：

choose. Color = TColor(BkgndColor);

执行一个颜色选择对话框:

TChooseColorDialog(this, choose, 0, "颜色选择对话框"). Execute();

根据返回值设置窗口的背景颜色并重绘窗口:

SetBkgndColor(choose. Color. Rgb());

Invalidate();

CmFont 函数用来响应菜单项命令"选择字体",初始化字体对话框数据 FontData:

```
FontData. Flags = CF_EFFECTS | CF_FORCEFONTEXIST | CF_SCREENFONTS;
FontData. Color = TColor::Black;
FontData. FontType = SCREEN_FONTTYPE;
FontData. SizeMin = 0;
FontData. SizeMax = 0;
```

执行字体选择对话框,并根据获得的字体重绘窗口:

```
if (TChooseFontDialog(this, FontData, 0, "字体选择框"). Execute() == IDOK)
  {
      delete Font;
      Font = new TFont(&FontData. LogFont);
      IsDirty = true;
      Invalidate();
  }
```

CmPrint 函数响应菜单项命令"打印",它构造了一个 TPrintStr 类的对象,并调用 TPrinter::Print 函数实现打印,程序会自动调用打印对话框。

```
void TcomdlgWindow::CmPrint ()
{
  if (Printer)
  {
      TPrintStr printout(this, FileData. FileName);
      printout. SetBanding(TRUE);
      Printer->Print(this, printout, TRUE);
  }
}
```

CmPntSet 函数响应菜单项命令"打印设置",提供调用 TPrinter::Setup 函数打开打印设置对话框,让用户设置打印参数。

```
void TcomdlgWindow::CmPntSet ()
{
  if (Printer)
      Printer->Setup(this);
}
```

CmString 函数响应菜单项命令"字符串输入",它执行了一个字符串输入对话框,让用

户输入要显示或打印的字符串。然后重绘窗口。

```
void TcomdlgWindow::CmString ()
{
  if ((TInputDialog(this, "输入框", "请输入一个字符串:",
              Str, sizeof(Str))).Execute() == IDOK)
  {
      Invalidate();
      IsDirty = true;
  }
}
```

TcomdlgWindow::Paint 函数用来在客户窗口中以特定的字体、颜色绘制特定的字符串。

```
void TcomdlgWindow::Paint (TDC& dc, bool erase, TRect& rect)
{
      TWindow::Paint(dc, erase, rect);
      if (Font)
      {
          dc.SetBkColor(TColor(BkgndColor));
          dc.SelectObject( * Font);
          dc.SetTextColor(FontData.Color);
      }
      dc.DrawText(Str, strlen(Str), GetClientRect(), DT_CENTER | DT_WORDBREAK);
}
```

CmOpen 函数响应菜单项命令“打开文件”，它先通过 CanClose 函数检查当前文件是否应该存盘，然后执行一个文件对话框，并获得用户输入的文件名，调用函数 OpenFile 来打开文件。

```
void TcomdlgWindow::CmOpen ()
{
  if (CanClose())
      if ((TFileOpenDialog(this, FileData, 0, "打开文件")).Execute() == IDOK)
          OpenFile();
}
```

TPrintout 类的派生类 TPrintStr 类提供了文件打印的方法，其中直接进行打印操作的函数是 PrintPage 函数。因为程序中只打印一页，所以其它的函数都比较简单。

```
class TPrintStr : public TPrintout
{
public:
    TPrintStr (TcomdlgWindow * Window, const char far * title);
    void GetDialogInfo(int& minPage, int& maxPage,
                       int& selFromPage, int& selToPage);
```

```
        void PrintPage(int page, TRect& rect, unsigned flags);
        void SetBanding(BOOL b) {Banding = b;}
        BOOL HasPage(int pageNumber) {return pageNumber == 1;}
    TcomdlgWindow * Window;
    };      //{{TPrintStr}}
```

PrintPage 函数用来打印文件，它调用了 TcomdlgWindow 窗口类的成员函数 Paint，使得打印的效果和在窗口中显示的效果相同，当然在不同的程序中打印的方法也不同，你不必一定要调用窗口类的成员函数 Paint。

```
void TPrintStr::PrintPage(int, TRect& rect, unsigned)
{
  int prevMode;
  TSize oldVExt, oldWExt;
  ......
  // Call the window to paint itself
  Window->Paint( * DC, FALSE, rect);
  ......
}
```

图 4-17～图 4-20 所示为运行通用对话框程序的结果，图 4-21 所示为通用对话框的项目文件。

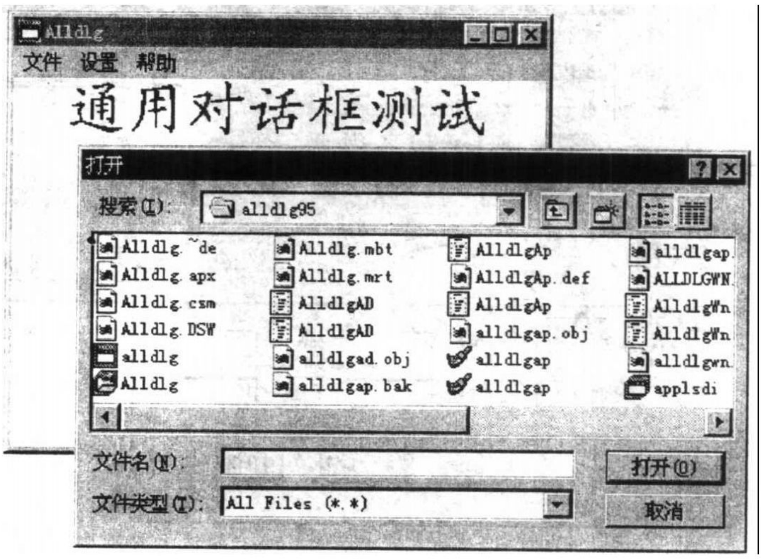


图 4-17　运行通用对话框程序(一)

下面是通用对话框应用程序的部分代码：

```
//----------------------------------------------------------
//  FILE:            AlldlgAp.h
//  Class definition for TAlldlgApp (TApplication).
//----------------------------------------------------------
#if !defined(alldlgap_h)            // Sentry, use file only if it's not already included.
#define alldlgap_h
```

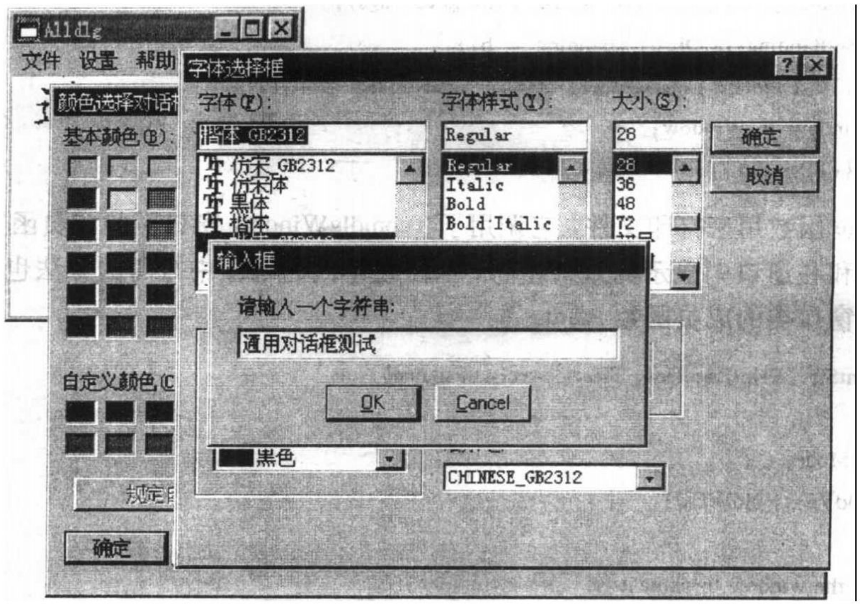


图 4-18　运行通用对话框程序(二)

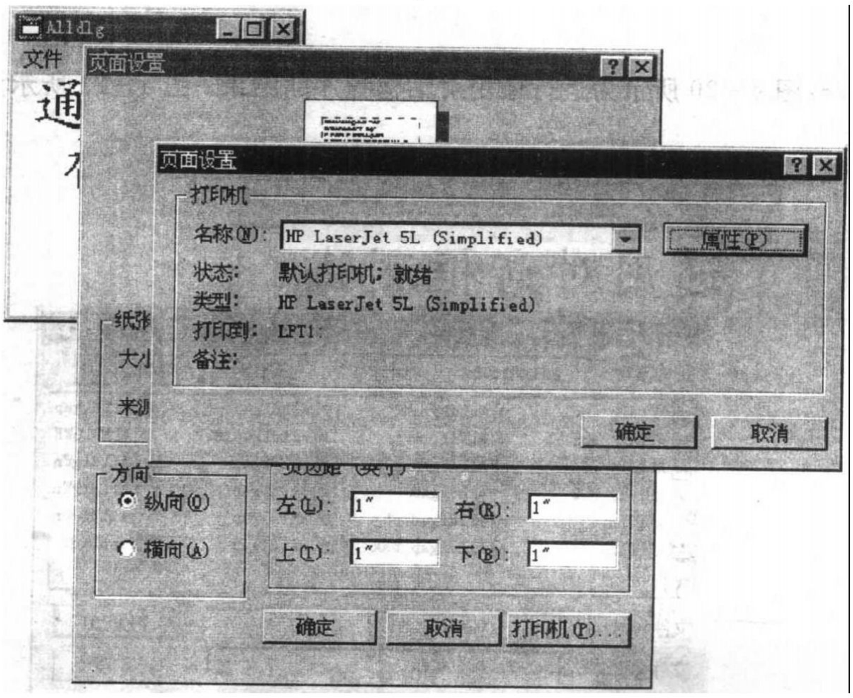


图 4-19　运行通用对话框程序(三)

```
#include <owl/opensave.h>
#include "AlldlgAp.rh"                    // Definition of all resources.
//
// FrameWindow must be derived to override Paint for Preview and Print.
//
//{{TDecoratedFrame = TSDIDecFrame}}
class TSDIDecFrame : public TDecoratedFrame {
  public:
```

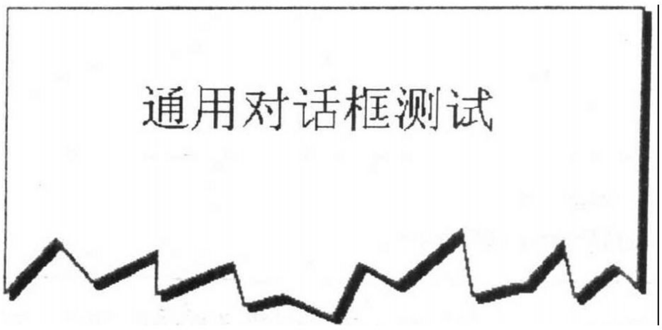


图 4-20 通用对话框程序完成的打印样张

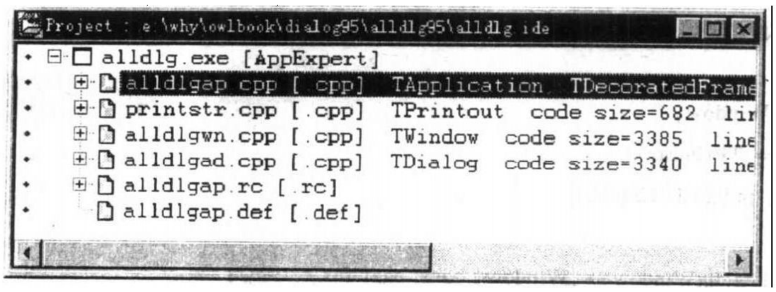


图 4-21 通用对话框项目文件

```
        TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool trackMenuSe-
lection = false, TModule * module = 0);
        ~TSDIDecFrame();
};      //{{TSDIDecFrame}}

//{{TApplication = TAlldlgApp}}
class TAlldlgApp : public TApplication {
  private:
  public:
    TAlldlgApp();
    virtual ~TAlldlgApp();
    TOpenSaveDialog::TData FileData;                // Data to control open/saveas standard dialog.
    void OpenFile();
//{{TAlldlgAppVIRTUAL_BEGIN}}
  public:
    virtual void InitMainWindow();
//{{TAlldlgAppVIRTUAL_END}}

//{{TAlldlgAppRSP_TBL_BEGIN}}
  protected:
    void CmFileNew();
    void CmFileOpen();
    void CmHelpAbout();
//{{TAlldlgAppRSP_TBL_END}}
```

```
DECLARE_RESPONSE_TABLE(TAlldlgApp);
};     //{{TAlldlgApp}}
#endif     // alldlgap_h sentry.

//----------------------------------------------------------------
//  FILE:          printstr.h
//  Class definition for TPrintStr (TPrintout).
//----------------------------------------------------------------
#if ! defined(printstr_h)          // Sentry, use file only if it's not already included.
#define printstr_h
#include <owl/printer.h>
#include "AlldlgAp.rh"                  // Definition of all resources.

//class TcomdlgWindow;
//{{TPrintout = TPrintStr}}
class TPrintStr : public TPrintout {
  public:
    TPrintStr(TAlldlgWindow *_Window, const char far * title);
    virtual ~TPrintStr();
public:
    TAlldlgWindow * Window;

//{{TPrintStrVIRTUAL_BEGIN}}
  public:
    virtual void PrintPage(int page, TRect& rect, uint flags);
    virtual void GetDialogInfo(int& minPage, int& maxPage, int& selFromPage, int& selToPage);
    void SetBanding(bool b)
    {
      Banding = b;
    }
    bool HasPage(int pageNumber)
    {
      return pageNumber == 1;
    }
//{{TPrintStrVIRTUAL_END}}
};     //{{TPrintStr}}
#endif     // printstr_h sentry.

//----------------------------------------------------------------
//  FILE:          AlldlgWn.h
//  Class definition for TAlldlgWindow (TWindow).
//----------------------------------------------------------------
#if ! defined(alldlgwn_h)          // Sentry, use file only if it's not already included.
```

```
#define alldlgwn_h
#include <owl/choosefo.h>
#include <owl/printer>
#include "AlldlgAp.rh"                      // Definition of all resources.

//{{TWindow = TAlldlgWindow}}
class TAlldlgWindow : public TWindow {
  public:
    TAlldlgWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TAlldlgWindow();
    TFont * Font;
    TChooseFontDialog::TData FontData;
    char Str[20];
    TPrinter * Printer;

//{{TAlldlgWindowVIRTUAL_BEGIN}}
  public:
    virtual void Paint(TDC& dc, bool erase, TRect& rect);
//{{TAlldlgWindowVIRTUAL_END}}

//{{TAlldlgWindowRSP_TBL_BEGIN}}
  protected:
   void CmString();
   void CmFont();
   void CmColor();
   void EvSize(uint sizeType, TSize& size);
   void CmPrnset();
   void CmPrint();
//{{TAlldlgWindowRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TAlldlgWindow);
};     //{{TAlldlgWindow}}
#endif     // alldlgwn_h sentry.

//----------------------------------------------------------------------
//  FILE:          AlldlgAp.cpp//  Source file for implementation of TAlldlgApp (TApplication).
//----------------------------------------------------------------------
#include <owl/pch.h>
#include <stdio.h>
#include "AlldlgAp.h"
#include "AlldlgWn.h"                   // Definition of client class.
#include "AlldlgAD.h"                   // Definition of about dialog.

//{{TAlldlgApp Implementation}}
```

```
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TAlldlgApp, TApplication)
//{{TAlldlgAppRSP_TBL_BEGIN}}
  EV_COMMAND(CM_FILENEW, CmFileNew),
  EV_COMMAND(CM_FILEOPEN, CmFileOpen),
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
//{{TAlldlgAppRSP_TBL_END}}
END_RESPONSE_TABLE;
//-----------------------------------
// TAlldlgApp
//
TAlldlgApp::TAlldlgApp() : TApplication("Alldlg")
{
// Common file file flags and filters for Open/Save As dialogs. Filename and directory are
  // computed in the member functions CmFileOpen, and CmFileSaveAs.
  //
  FileData.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT;
  FileData.SetFilter("All Files (*.*)|*.*|");
  FileData.DefExt = "txt";

#if defined(BI_PLAT_WIN32)
  if (TSystem::IsNT() || TSystem::IsWin95())
    FileData.Flags |= OFN_LONGNAMES;

  if (TSystem::IsWin95())
    FileData.Flags |= OFN_EXPLORER;
#endif
}
TAlldlgApp::~TAlldlgApp()
{
}
//-----------------------------------
// TAlldlgApp
// ~~~~~
// Application intialization.
//
void TAlldlgApp::InitMainWindow()
{
  if (nCmdShow != SW_HIDE)
    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;
```

```
  TSDIDecFrame * frame = new TSDIDecFrame(0, GetName(), 0, false);

  // Assign ICON w/ this application.
  //
  frame->SetIcon(this, IDI_SDIAPPLICATION);
  frame->SetIconSm(this, IDI_SDIAPPLICATION);
// Menu associated with window and accelerator table associated with table.
  //
  frame->AssignMenu(IDM_SDI);
// Associate with the accelerator table.
  //
  frame->Attr.AccelTable = IDM_SDI;
SetMainWindow(frame);

  frame->SetMenuDescr(TMenuDescr(IDM_SDI));
}
//--------------------------------------------------------
// TAlldlgApp
// ~~~~~~~~~
// Menu File New command
//
void TAlldlgApp::CmFileNew()
{
  GetMainWindow()->MessageBox("新建文件","消息");
}
//--------------------------------------------------------
// TAlldlgApp
// ~~~~~~~~~
// Menu File Open command
//
void TAlldlgApp::CmFileOpen()
{
  // Display standard Open dialog box to select a file name.
  //
  *FileData.FileName = 0;
TAlldlgWindow * client = TYPESAFE_DOWNCAST(GetMainWindow()->GetClientWindow(), TAlldl-
gWindow); // Client window for the frame.
  if (client->CanClose())
    if (TFileOpenDialog(GetMainWindow(), FileData).Execute() == IDOK)
      OpenFile();
}
void TAlldlgApp::OpenFile()
```

```
{
  GetMainWindow()->MessageBox("新建文件","消息");
}
//{{TSDIDecFrame Implementation}}
TSDIDecFrame::TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool
trackMenuSelection, TModule * module)
:    TDecoratedFrame(parent, title, !clientWnd ? new TAlldlgWindow(0, "") : clientWnd, trackMenuSe-
lection, module)
{
}
TSDIDecFrame::~TSDIDecFrame()
{
}
//--------------------------------------------------------
// TAlldlgApp
// ~~~~~~~~~
// Menu Help About Alldlg command
//
void TAlldlgApp::CmHelpAbout()
{
  TAlldlgAboutDlg(MainWindow).Execute();
}
int OwlMain(int , char * [])
{
  TAlldlgApp app;
  return app.Run();
}

//--------------------------------------------------------
//  FILE:          printstr.cpp
//  Source file for implementation of TPrintStr (TPrintout).
//--------------------------------------------------------
#include <owl/pch.h>
#include "AlldlgWn.h"
#include "printstr.h"

//{{TPrintStr Implementation}}
TPrintStr::TPrintStr(TAlldlgWindow *_Window, const char far * title)
:      TPrintout(title)
{
    Window=_Window;
}
TPrintStr::~TPrintStr()
```

```
{
}
void TPrintStr::PrintPage(int page, TRect& rect, uint flags)
{
  TPrintout::PrintPage(page, rect, flags);

  int prevMode;
  TSize oldVExt, oldWExt;
      prevMode = DC->SetMapMode(MM_ISOTROPIC);
      TRect windowSize = Window->GetClientRect();
      DC->SetViewportExt(PageSize, &oldVExt);
      DC->SetWindowExt(windowSize.Size(), &oldWExt);
      DC->IntersectClipRect(windowSize);
      DC->DPtoLP(rect, 2);

  // Call the window to paint itself
  Window->Paint(*DC, FALSE, rect);

  // Restore changes made to the DC
      DC->SetWindowExt(oldWExt);
      DC->SetViewportExt(oldVExt);
      DC->SetMapMode(prevMode);
}
void TPrintStr::GetDialogInfo(int& minPage, int& maxPage, int& selFromPage, int& selToPage)
{
  TPrintout::GetDialogInfo(minPage, maxPage, selFromPage, selToPage);

  minPage = 0;
  maxPage = 0;
  selFromPage = selToPage = 0;
}

//----------------------------------------------------------------------
// FILE:          AlldlgWn.cpp
// Source file for implementation of TAlldlgWindow (TWindow).
//----------------------------------------------------------------------
#include <owl/pch.h>
#include <owl/inputdia.h>
#include <owl/chooseco.h>
#include <stdio.h>
#include "AlldlgAp.h"
#include "AlldlgWn.h"
#include "printstr.h"
```

```
//
// Build a response table for all messages/commands handled by the application.
//
DEFINE_RESPONSE_TABLE1(TAlldlgWindow, TWindow)
//{{TAlldlgWindowRSP_TBL_BEGIN}}
  EV_COMMAND(CM_STRING, CmString),
  EV_COMMAND(CM_FONT, CmFont),
  EV_COMMAND(CM_COLOR, CmColor),
  EV_WM_SIZE,
  EV_COMMAND(CM_PRNSET, CmPrnset),
  EV_COMMAND(CM_PRINT, CmPrint),
//{{TAlldlgWindowRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{TAlldlgWindow Implementation}}

//----------------------------------------
// TAlldlgWindow
// ~~~~~~~~
// Construction/Destruction handling.
//
TAlldlgWindow::TAlldlgWindow(TWindow * parent, const char far * title, TModule * module)
:       TWindow(parent, title, module)
{
      SetBkgndColor(RGB(255,255,255));
      Font = 0;
      sprintf(Str,"通用对话框测试");
      Printer = new TPrinter;
}
TAlldlgWindow::~TAlldlgWindow()
{
  Destroy();
  delete Printer;
}
void TAlldlgWindow::CmString()
{
  if ((TInputDialog(this, "输入框",
                                "请输入一个字符串:",
                                Str,
                                sizeof(Str))).Execute() == IDOK)
  Invalidate();
}
```

```
void TAlldlgWindow::Paint(TDC& dc, bool erase, TRect& rect)
{
  TWindow::Paint(dc, erase, rect);

  if (Font)
  {
    dc.SetBkColor(TColor(BkgndColor));
    dc.SelectObject( * Font);
    dc.SetTextColor(FontData.Color);
  }
  dc.DrawText(Str, strlen(Str), GetClientRect(), DT_CENTER | DT_WORDBREAK);
}
void TAlldlgWindow::CmFont()
{
  if (Font)
  {
      FontData.Flags |= CF_INITTOLOGFONTSTRUCT;
  }
  else
  {
      FontData.DC = 0;
      FontData.Flags = CF_EFFECTS | CF_FORCEFONTEXIST | CF_SCREENFONTS;
      FontData.Color = TColor::Black;
      FontData.Style = 0;
      FontData.FontType = SCREEN_FONTTYPE;
      FontData.SizeMin = 0;
      FontData.SizeMax = 0;
  }
  if (TChooseFontDialog(this, FontData, 0, "字体选择框").Execute() == IDOK)
  {
      delete Font;
      Font = new TFont(&FontData.LogFont);
      Invalidate();
  }
}
void TAlldlgWindow::CmColor()
{
  TChooseColorDialog::TData choose;
  static TColor custColors[16] = {
      0x010101L, 0x101010L, 0x202020L, 0x303030L,
      0x404040L, 0x505050L, 0x606060L, 0x707070L,
      0x808080L, 0x909090L, 0xA0A0A0L, 0xB0B0B0L,
      0xC0C0C0L, 0xD0D0D0L, 0xE0E0E0L, 0xF0F0F0L
```

```
  };

  choose.Flags = CC_RGBINIT;
  choose.Color = TColor(BkgndColor);
  choose.CustColors = custColors;
  if (TChooseColorDialog(this, choose, 0, "颜色选择对话框").Execute() == IDOK)
  {
      SetBkgndColor(choose.Color.Rgb());
      Invalidate();
  }
}
void TAlldlgWindow::EvSize(uint sizeType, TSize& size)
{
      TWindow::EvSize(sizeType, size);
      Invalidate();
}
void TAlldlgWindow::CmPrnset()
{
  if (Printer)
      Printer->Setup(this);
}
void TAlldlgWindow::CmPrint()
{
  if (Printer)
  {
      TPrintStr printout(this, "ABC");
      printout.SetBanding(TRUE);
      Printer->Print(this, printout, TRUE);
  }
}
```

4.23 多页对话框程序实例

可切换的对话框你一定见过，比如 MS Word 6.0 中的“选项”对话框。在 Borland C++ 5.0中也有许多可以切换的对话框，最常用的是 Project Options 对话框，当我们在 Topics 中点击一个项目的时候，对话框右半部分的内容就会随之改变。这样，从一个对话框中可以获得和设置的信息就多了几倍。

使用 OWL 实现这样的对话框并不复杂，因为我们可以通过控件对象来方便地操作对话框中的各个控件，使它们成为“可见”或者“不可见”的(通常对话框中的控件都是可见的)，我们还可以将控件的位置放到对话框中或者对话框外(通常对话框中的控件都是在对话框内)。

“显示\隐藏”法是比较方便的方法。我们在资源编辑器中将所有的控件都拖动到它们

的位置上(完成这一步的具体方法有很多，一般采用块移动的方法比较方便)，这时对话框中显得非常杂乱，许多控件互相覆盖(见图 4-22)，剩下的事由程序来完成。当用户按下某一幕的按钮时，对话框把这一幕的控件都设置为“可见的”，而将其它幕的控件都设置为“不可见的”，这样，在对话框中将只显示一个场景。

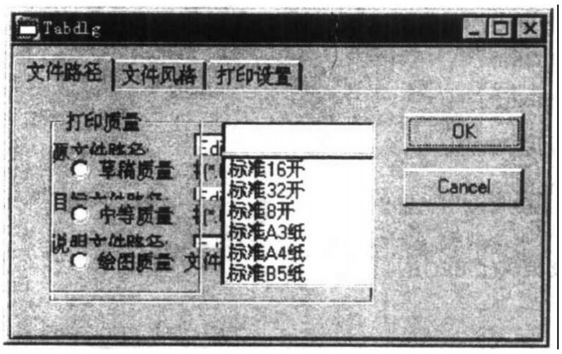


图 4-22　用"显示\隐藏"的方法实现对话框的切换

如何使用“移动位置”的方法来实现对话框的切换呢？像编辑普通对话框一样，我们可以在资源编辑器中编辑对话框，所不同的是对话框中需要有接收切换命令的控件，比如，可采用按钮“第一幕”、“第二幕”等；还要将要显示的控件分别放到对话框的内部或外部，比如，将第二幕和第三幕所需的控件放到对话框外部，而将第一幕事先放到对话框的内部，因为对话框显示的时候，初始化的设置就是第一幕。这样，当用户按下“第二幕”按钮时，对话框对象将处理这个事件，它将第一幕中的控件一个一个地移动到对话框外部，并将第二幕的控件一个一个地移动到对话框中。这时，对话框中显示的场景就是第二幕。实际上，在一个场景显示的时候，其它场景的控件都是存在的，只是你无法按动它们，因为它们在对话框外。你知道，将一个控件从一个位置移动到另外一个位置要比将它隐藏或显示的操作要麻烦得多，所以下面给出的程序中是以“显示\隐藏”方式切换控件的。

在程序中我们选择了通用控件 TTabControl 作为对话框切换命令的输入控件，Common 中的通用控件可以使应用程序增色不少。

下面就是程序实例。如前所述，它使用“显示\隐藏”方式实现可切换对话框，并采用 TTabControl 控件作为切换的消息来源。与前面的方法一样，我们先使用 AppExpert 创建一个基于对话框的应用程序框架；再使用对话框编辑器在作为客户窗口的对话框中加入一系列控件，包括 TTabControl 控件；最后，使用 ClassExpert 对 TTabdlgDlgClient 类进行编辑，定义它所包括的控件对象，具体定义如下：

```
class TTabdlgDlgClient : public TDialog {
  public:
    TTabdlgDlgClient(TWindow * parent, TResId resId = IDD_CLIENT, TModule * module = 0);
    virtual ~TTabdlgDlgClient();
  protected:
    TStatic * Static1;
    TStatic * Static2;
    TStatic * Static3;
    TComboBox * ComboBox;
```

```
        TEdit * Edit1;
        TEdit * Edit2;
        TEdit * Edit3;
        TGroupBox * GroupBox1;
        TGroupBox * GroupBox2;     TRadioButton * RadioButton1;
        TRadioButton * RadioButton2;
        TRadioButton * RadioButton3;
        TRadioButton * RadioButton4;
        TRadioButton * RadioButton5;
        TRadioButton * RadioButton6;
        TTabControl * tab;
//{{TTabdlgDlgClientVIRTUAL_BEGIN}}
    public:
        virtual void SetupWindow();
//{{TTabdlgDlgClientVIRTUAL_END}}
    void ShowCtl(int nActiveTab);
//{{TTabdlgDlgClientRSP_TBL_BEGIN}}
    protected:
    void TabSelChange(TNotify far&);
//{{TTabdlgDlgClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TTabdlgDlgClient);
};      //{{TTabdlgDlgClient}}
```

其中，tab 指向一个 TTabControl 对象。重载的 SetupWindow 函数对对话框和其中的控件进行必要的初始化，包括给组合框控件增加内容、给 tab 增加项目以及调用 ShowCtl 函数显示这些控件。

在事件响应表中定义了由 TabSelChange 函数来响应 TTabControl 控件变化事件：

```
DEFINE_RESPONSE_TABLE1(TTabdlgDlgClient, TDialog)
    EV_TCN_SELCHANGE(IDC_SYSTABCONTROL1, TabSelChange),
END_RESPONSE_TABLE;
```

TabSelChange 函数对用户选择的页进行判断后调用 ShowCtl 显示控件：

```
void TTabdlgDlgClient::TabSelChange(TNotify far& /* nm */)
{
    int nActiveTab;
    nActiveTab = tab->GetSel();
    ShowCtl(nActiveTab);
}
```

ShowCtl 函数的参数 nActiveTab 传输了用户选择的页码，通过对控件的显示属性进行设置(SW_SHOW 或是 SW_HIDE)来显示或隐藏控件，从而实现多页之间的切换：

```
void TTabdlgDlgClient::ShowCtl(int nActiveTab)
{
```

```
    Static1->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    Static2->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    Static3->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    ComboBox->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
    Edit1->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    Edit2->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    Edit3->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
    GroupBox1->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
    GroupBox2->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
    RadioButton1->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
    RadioButton2->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
    RadioButton3->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
    RadioButton4 ->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
    RadioButton5->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
    RadioButton6 ->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
}
```

图 4-23～图 4-25 分别给出多页对话框的“第一幕”、“第二幕”及“第三幕”，图 4-26 中示出了多页对话框程序的项目文件。

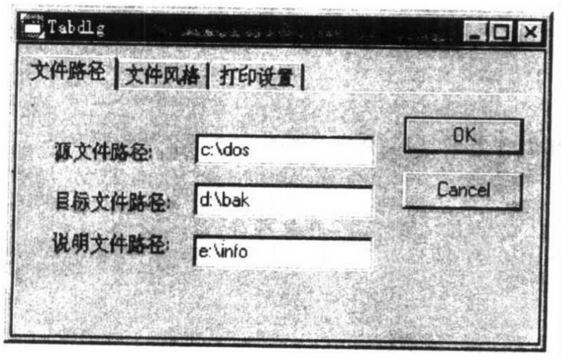


图 4-23　对话框第一幕

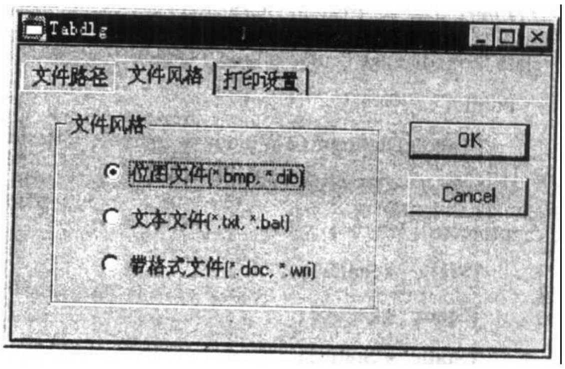


图 4-24　对话框第二幕

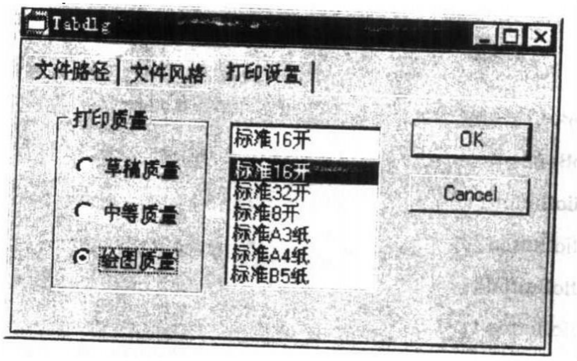


图 4-25　对话框第三幕

下面是多页对话框应用程序的部分代码：

```
//------------------------------------------------------------
//  FILE:           TbdlgDlC.h
```

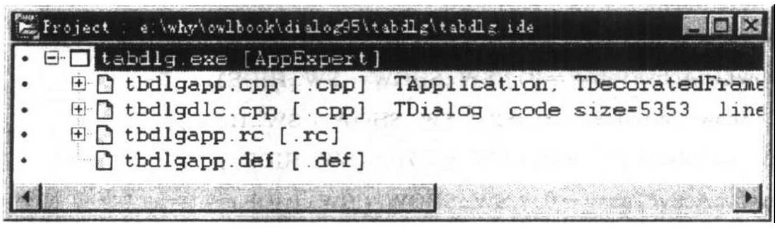


图 4-26　多页对话框项目文件

```
//  Class definition for TTabdlgDlgClient (TDialog).
//-----------------------------------------------------------------
#if !defined(tbdlgdlc_h)              // Sentry, use file only if it's not already included.
#define tbdlgdlc_h
#include "TbdlgApp.rh"                   // Definition of all resources.
#include <owl/radiobut.h>
#include <owl/groupbox.h>
#include <owl/edit.h>
#include <owl/combobox.h>
#include <owl/tabctrl.h>

//{{TDialog = TTabdlgDlgClient}}
class TTabdlgDlgClient : public TDialog {
  public:
    TTabdlgDlgClient(TWindow* parent, TResId resId = IDD_CLIENT, TModule* module = 0);
    virtual ~TTabdlgDlgClient();
  protected:
    TStatic  *Static1;
    TStatic  *Static2;
    TStatic  *Statio3;
    TComboBox  *ComboBox;
    TEdit  *Edit1;
    TEdit  *Edit2;
    TEdit  *Edit3;
    TGroupBox  *GroupBox1;
    TGroupBox  *GroupBox2;
    TRadioButton  *RadioButton1;
    TRadioButton  *RadioButton2;
    TRadioButton  *RadioButton3;
    TRadioButton  *RadioButton4;
    TRadioButton  *RadioButton5;
    TRadioButton  *RadioButton6;
    TTabControl* tab;
//{{TTabdlgDlgClientVIRTUAL_BEGIN}}
  public:
```

```
    virtual void SetupWindow();
//{{TTabdlgDlgClientVIRTUAL_END}}
   void ShowCtl(int nActiveTab);
//{{TTabdlgDlgClientRSP_TBL_BEGIN}}
   protected:
   void TabSelChange(TNotify far&);
//{{TTabdlgDlgClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TTabdlgDlgClient);
};      //{{TTabdlgDlgClient}}
#endif      // tbdlgdlc_h sentry.

//----------------------------------------------------------------------------
//  FILE:               TbdlgDlC.cpp
//  Source file for implementation of TTabdlgDlgClient (TDialog).
//----------------------------------------------------------------------------
#include <owl/pch.h>
#include "TbdlgApp.h"
#include "TbdlgDlC.h"

const char * TabTitle[] = { "文件路径", "文件风格", "打印设置"};
//
// Build a response table for all messages/commands handled by the application.
//
DEFINE_RESPONSE_TABLE1(TTabdlgDlgClient, TDialog)
//{{TTabdlgDlgClientRSP_TBL_BEGIN}}
  EV_TCN_SELCHANGE(IDC_SYSTABCONTROL1, TabSelChange),
//{{TTabdlgDlgClientRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{TTabdlgDlgClient Implementation}}

//---------------------------------------------------------------------
// TTabdlgDlgClient
// ~~~~~~~~
// Construction/Destruction handling.
//
TTabdlgDlgClient::TTabdlgDlgClient(TWindow * parent, TResId resId, TModule * module)
:  TDialog(parent, resId, module)
{
    Static1 = new TStatic(this, IDC_STATIC1, 255);
    Static2 = new TStatic(this, IDC_STATIC2, 255);
    Static3 = new TStatic(this, IDC_STATIC3, 255);
    ComboBox = new TComboBox(this, IDC_COMBOBOX1, 1);
```

```
    Edit1 = new TEdit(this, IDC_EDIT1, 255);
    Edit2 = new TEdit(this, IDC_EDIT2, 255);
    Edit3 = new TEdit(this, IDC_EDIT3, 255);
    GroupBox1 = new TGroupBox(this, IDC_GROUPBOX1);
    GroupBox2 = new TGroupBox(this, IDC_GROUPBOX2);
    RadioButton1 = new TRadioButton(this, IDC_RADIOBUTTON1, 0);
    RadioButton2 = new TRadioButton(this, IDC_RADIOBUTTON2, 0);
    RadioButton3 = new TRadioButton(this, IDC_RADIOBUTTON3, 0);
    RadioButton4 = new TRadioButton(this, IDC_RADIOBUTTON4, 0);
    RadioButton5 = new TRadioButton(this, IDC_RADIOBUTTON5, 0);
    RadioButton6 = new TRadioButton(this, IDC_RADIOBUTTON6, 0);
    tab = new TTabControl(this, IDC_SYSTABCONTROL1);
}

void TTabdlgDlgClient::TabSelChange(TNotify far& /* nm */)
{
  int nActiveTab;
  nActiveTab = tab->GetSel();
  ShowCtl(nActiveTab);
}

void TTabdlgDlgClient::ShowCtl(int nActiveTab)
{
  Static1->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  Static2->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  Static3->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  ComboBox->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
  Edit1->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  Edit2->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  Edit3->Show((nActiveTab==0)? SW_SHOW : SW_HIDE);
  GroupBox1->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
  GroupBox2->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
  RadioButton1->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
  RadioButton2->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
  RadioButton3->Show((nActiveTab==1)? SW_SHOW : SW_HIDE);
  RadioButton4 ->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
  RadioButton5->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
  RadioButton6 ->Show((nActiveTab==2)? SW_SHOW : SW_HIDE);
}

TTabdlgDlgClient::~TTabdlgDlgClient()
{
  Destroy();
```

```
}

void TTabdlgDlgClient::SetupWindow()
{
  TDialog::SetupWindow();
  for (int i=0; i<sizeof(TabTitle)/sizeof(TabTitle[0]); i++)
  {
    TTabItem tabItem(TabTitle[i]);
    tab->Add(tabItem);
  }
  ComboBox->AddString("标准 A4 纸");
  ComboBox->AddString("标准 A3 纸");
  ComboBox->AddString("标准 B5 纸");
  ComboBox->AddString("标准 16 开");
  ComboBox->AddString("标准 32 开");
  ComboBox->AddString("标准 8 开");
  ShowCtl(0);
}
```

4.24 信息框

信息框并不是对话框类，但它与对话框有一些相似之处(见图 4-27)。在 OWL 中显示信息框是通过调用 TWindow::MessageBox 函数实现的。

```
int MessageBox(const char far * text,
          const char far * caption = 0,
          uint type = MB_OK);
```

典型的函数使用方法是：

```
MessageBox("给用户一个信息",
          "使用信息框",
                    MB_OKCANCEL | MB_ICONINFORMATION);
```

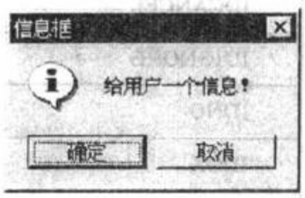


图 4-27 信息框给用户提供信息

函数中的参数包括了要显示的信息、信息框的标题和信息框的风格(缺省时为 OK 按钮)。信息框可以包括按钮和图标(以及它们的组合)，MB_Xxxx 结构说明了信息框的中按钮或图标的种类，其取值相应的意义见表 4-11。函数的返回值及其意义则见表 4-12。

表 4-11

取 值	意 义
MB_OK	在信息框中增加 OK 按钮
MB_OKCANCEL	在信息框中增加 Cancel 和 OK 按钮

续表

取　值	意　义
MB_ABORTRETRYIGNORE	在信息框中增加 Abort，Retry，和 Ignore 按钮
MB_YESNOCANCEL	在信息框中增加 Yes，No 和 Cancel 按钮
MB_YESNO	在信息框中增加 Yes 和 No 按钮
MB_RETRYCANCEL	在信息框中增加 Retry 和 Cancel 按钮
MB_ICONHAND	在信息框中增加停止图标
MB_ICONQUESTION	在信息框中增加问号图标
MB_ICONEXCLAMATION	在信息框中增加叹号图标
MB_ICONASTERISK	在信息框中增加一个小写字母 i
MB_ICONINFORMATION	在信息框中增加一个小写字母 i
MB_ICONSTOP	在信息框中增加停止图标
MB_DEFBUTTON1	将第一个按钮设置为缺省按钮
MB_DEFBUTTON2	将第二个按钮设置为缺省按钮
MB_DEFBUTTON3	将第三个按钮设置为缺省按钮

表 4-12

取　值	意　义
IDABORT	用户选择 ABORT 按钮
IDCANCEL	用户选择 CANCEL 按钮
IDIGNORE	用户选择 IGNORE 按钮
IDNO	用户选择 NO 按钮
IDOK	用户选择 OK 按钮
IDRETRY	用户选择 RETRY 按钮
IDYES	用户选择 YES 按钮

如果使用 BWCC，消息框就是 BWCC 风格的；如果使用 CTRL 3D，消息框就是 CTRL 3D 风格的。图 4-28～图 4-29 给出了不同风格的信息框：

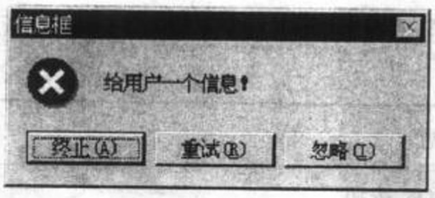


图 4-28　信息框

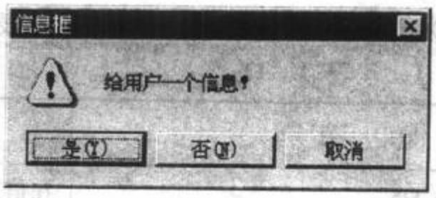


图 4-29　信息框

第 5 章

OWL 5.0 应用实例

用计算机对自然界进行模拟的技术是近年随着计算机技术的成熟而发展起来的。牛津大学的 Richard Dawkins 曾经通过用计算机模拟生物进化过程的形态演变来深入地认识生物进化。Dawkins 自己编写了一个模拟生物进化过程中形态演变的程序，程序所显示的生物是由计算机产生，这些生物与我们常见的生物类似。程序中，这些生物通过“人工选择”过程而进化。所谓“人工选择”就是计算机操作人员随机地从现有的生物形态的九种可能的形态中选择一种作为进化成未来几代生物形态的基础。Dawkins 的程序所产生的生物形态既稀奇古怪又十分有趣。但它们却不是自然地进化过来的，也就是说，它们不是在一种内在的选择压力下进化的。Dawkins 认为有可能编出一个计算机程序来模拟自然选择过程。

加利福尼亚州坦普尔市的一位教师 Michael Palmiter 编写了一个叫做“模拟进化”的程序，在屏幕中极小的原始动物(虫子)在屏幕上到处爬动，吞食着紫色的细菌。随着虫子一代一代的繁衍，我们可以观察到虫子的进化过程。Palmiter 的虫子肯定使我们向 Dawkins 所设想的目标更接近了一步。

下面我们编写了一个类似于“模拟进化”程序，取名为“生物圈三号”。

5.1　编写一个有趣的程序“生物圈三号”

这是一个用计算机对自然界现象进行模拟的程序，它包括对生物的遗传与变异、自然选择、物种进化、生物与环境和生物链等自然现象的模拟。所有这些方面的模拟都是通过一种小的原始动物(它的样子很象虫子)和一种原始植物(大概是一种单细胞的藻类)的生存过程实现的。为了方便，我们下面把这种原始动物叫做“虫子”，而把原始植物叫做“草”(见图 5 - 1)。通过计算机，我们让虫子在屏幕上四处爬行，它们是在觅食，它们可以吃那些慢慢地随机长出的草。虫子们看起来全是一样的，但它们的行为却不大相同。有的虫子在毫无目的地到处乱爬，不停地向一个方向拐弯，因此总是在原地打转，结果什么都吃不到，而另外一些虫子目的比较明确，甚至好象是具有一定的智慧，它们井井有条的搜索，并且经常能够吃到草。这个“生物”世界就是这么美妙！程序中所描述的这个生物乐园有一番与众不同的意义，要知道，在自然界要进行数万年的进化在这个生物圈中只需要两个小时。两个小时内，虫子们就从它们的祖先——一种行为没有条理的虫子进化成具有各种各样适应环境行为的虫子。

程序中的虫子生活在一个矩形的方框中，而草则不断地在方框中生长。虫子要爬动并

吃掉草才能维持生命。每吃掉一个草它就能获得40个单位的能量，此能量足够虫子作40次移动(虫子每次移动需要消耗一个单位的能量)。在草比较多的地方，一只虫子在几分钟内就可以获得大量的能量。

另外一方面，如果一只虫子很长时间内都找不到什么草吃，那么它的能量积蓄就会逐渐变成0，这时，虫子就无精打采地停留在原地，仿佛它的末日即将来临，再过一会儿，它果然从屏幕上消失了，它死了。

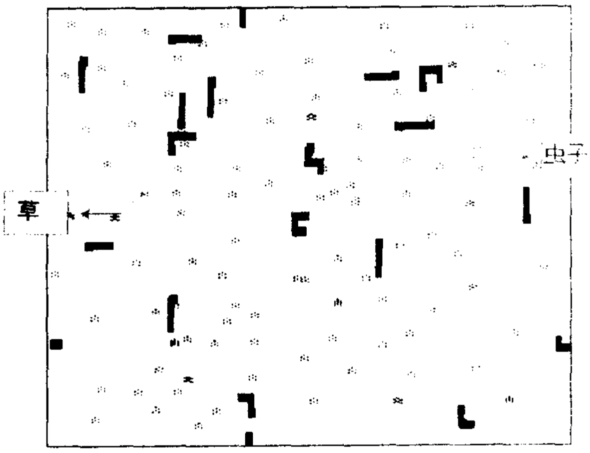


图 5-1 “生物圈三号”中的生物

对“生物圈三号”程序的达尔文主义的解释尽管抽象，但始终是围绕着控制虫子移动方式的“基因”进行的。在真正的原始动物中，这种特殊的基因可能并不存在，但是在程序中所产生的虫子有两个这样的基因：向左概率基因和向右概率基因。这已经足够控制虫子的运动方式了。我们设虫子下一步的运动方向有三种：向前、向左和向右，而三种方式是以一定的概率随机选择的，也就是说这三种概率的和是100%。那么如果我们确定其中的两种，比如向左的概率和向右的概率，我们就可以算出向前的概率。在程序中我们将虫子的基因封装在一个数据结构中：

```
struct TWormGene
{
int LeftProb;
int RightProb;
};
```

这些方向都是以虫子的观察点为基准的，虫子每作一次移动，其方向是按彩票摸奖那样的方式确定的，即从一个口袋中随机的取出一个签来。例如，如果程序摸到L，虫子就向左转弯。摸到某一个特定方向的概率是由赋于响应基因的一个数值决定的。因此，某个基因的值越高，该基因对虫子的总的移动方式所引起的作用就越大。例如，如果虫子的向左基因的值较大，那么虫子的大部分时间都是向左转的。

基因的每种可能的组合型式都对应一种移动方式，而且，一旦虫子的移动基因构成型式确定下来，便终生保持不变。也就是说：虫子只能希望它的后代超过自己。

一只虫子作10次移动后便进入了“成年期”(太早熟了！幸好只是虫子)，准备繁殖后代了。但是，只有处于强壮状态的虫子——也就是在电子生物膜内存储了1000单位以上的能

量的虫子——才能进行繁殖。草履虫的繁殖过程是两个生殖细胞的结合，但程序中的虫子都是按分裂的方式繁殖，即一个强壮的成年虫子一分为二成为两个新的个体，每一个虫子的能量是其母体能量的一半。新的虫子产生后，继承了母体的基因，但同时也出现了一定程度的变异：每个后代的某个基因的值略有升高和降低。

例如，假设有一只强壮的成年虫子的基因分别是

```
LeftProb=20;
RightProb=30;
```

它的两个子代(分别表示为A和B)可能具有下列经过变异的基因组合：

```
A:   LeftProb=22;
     RightProb=27;
B:   LeftProb=18;
     RightProb=28;
```

我们可以看到，通过变异子代A的向左拐弯基因的值增大了2%，而子代B的向右拐弯基因减小了2%。也就是说，子代A向左拐弯的趋势比其母体稍强，而子代B作向右拐弯的趋势比其母体稍弱。这些运动趋势的细微改变从计算机显示屏幕上观察到需要一定的时间和耐心。

最简单的"生物圈三号"的程序开始运行时，给一些虫子各赋一种随机选定的基因组合，这就使大多数虫子以一种无法预料的方式从矩形的一边到矩形的另外一边紧张地跳动。一般来说，刚开始虫子的死亡率很高，它们通常很快吃光周围的草，然后继续在这一片不毛之地上跳动，直到被饿死。当然，还有一些虫子幸存了下来。

时间一分一分地过去，虫子就这样一代一代地出现。一场引人入胜的生死斗争就在计算机中进行，几分钟后它的戏剧效果将真正达到高潮。这时，我们可以察觉到有一些虫子的行为已发生了改变，它们不是颤抖而是打滚，再过几十分钟将出现作短距离滑行移动的虫子，这些虫子的运动方式看来比它们的那些只会颤抖的祖先要高明得多。的确，正是因为这个原因，这个种群开始繁衍起来。

到一定时候，就会出现一些进行"巡航"的虫子，它们多数时间向前移动，但也不时地转一个弯，这就意味着它们几乎总是朝前爬行到草最密集的地方。这种行为只要在不多的虫子身上出现，很快就在整个群体中占主宰地位，因为这些"巡洋舰"一样的虫子最终会吃掉大部分草。"巡洋舰"种群的两个基因的取值通常都比较小(小于10%)，这就是说它们向前的趋势比较大(大于80%)，它们总是向前走几十步然后向左或向右拐一个弯。

尽管"巡洋舰"们构成了一类物种，然而在这个物种内部仍然具有不同的品种。例如，有些"巡洋舰"右转弯的时候比左转弯的次数多，当然还有偶尔作倒退的移动。有些"巡洋舰"大量繁殖出适应能力差的不良后代。一种最常见的遗传病是"盘旋病"，患有这种基因遗传病的虫子总是向一个方向转弯。这类虫子的两个基因的取值发生的不好的变异，其中一个基因值(比如向左的基因值为30%)比另外一个基因值(比如向右的基因值为5%)大得多，这样它们就总是向一个方向拐弯，这类不幸的虫子一般都是还没有尝到分裂繁殖的乐趣就一命呜呼了。图5-2中给出了"生物圈三号"中几种新物种。

观察"巡洋舰"这个种群的出现已经是十分有趣的事情了，但程序中还不只这些。如果

情况出现了变化，那么虫子将如何进化呢？为了回答这个问题，程序中设置了一个区域叫做"伊甸园"(见图 5-3)，在这个区域中放置特别多的草，而且草的补充率远远高出其它区域。

虫子一代一代的产生，又一代一代的消失。"巡洋舰"种群的虫子如前面所讲的那样不断地进化。但是在伊甸园内却出现了完全不同的一种情况。少数几只"跑得不快"的虫子幸运地进入了伊甸园，它们的那种毫无条理的觅食方式此时此地却帮了它们的大忙，依仗这种觅食方式，它们可以在伊甸园内游来游去，不管它们走到哪里，它们都能大吃一顿。

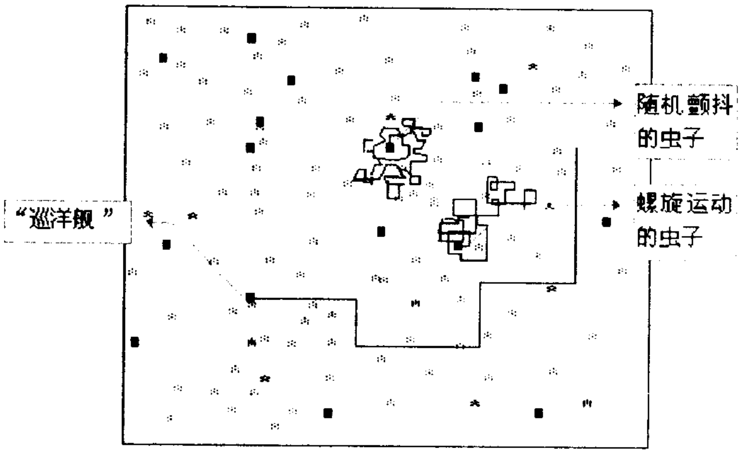


图 5-2　在"生物圈三号"中出现的几个新的物种

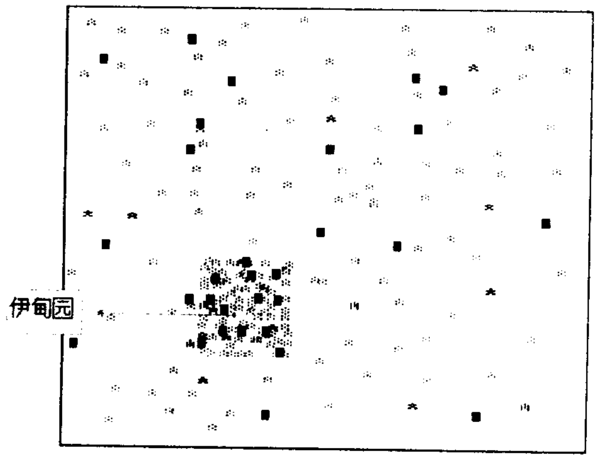


图 5-3　"生物圈三号"中的伊甸园

然而，随着伊甸园内的草越来越少，一种不利的环境压力开始起作用。靠晃动来捞取食物的虫子很快就发现它们的觅食方式不灵了，此时正是"盘旋者"这类虫子大显身手的时机，盘旋病通常被看作是灾难性的遗传病，但在"虫口过剩"的伊甸园中这个缺点就变成了优点。的确，经过一段时间以后，伊甸园中就只剩下了那些喜欢向一个方向转弯的虫子了。其原因很明显，如果虫子在伊甸园内频繁地向同一个方向转弯，那么它就总能够在伊甸园得到大量繁殖的机会。

最多只要几个小时，伊甸园就几乎完全被高度成熟的"盘旋者"所独占，我们不妨把它们称为"轨道飞行器"，它们沿一个特定的轨道转上许多圈，然后突然向外一跳，又在另外

一个轨道上运动，并且每一次都把路上的草捕食干净。

通过观察程序的运行我们可以发现，虫子们一代比一代聪明，而实际上在程序中，基因的变异是双向的(可以往先进的方式转变，同时也可以出现“返祖现象”)，变化的方式完全是随机的，但是在自然选择的情况下，生物圈毫不留情地淘汰了那些不能适应环境的虫子。尤其是在食物短缺时，大量的虫子因为饥饿而死亡，只有那些具有先进觅食方式的虫子才能活下来并且繁殖后代。这样的选择使得虫子的行为逐渐具有了一定的规范。

另外，程序可以控制生物圈中草的增长率和虫子吃草的能量增长率。但是我们可以发现不管这两个值如何变化，草的数目和虫子的数目总是满足一定的关系：当草的数目增加时，虫子的捕食效率增加，在优越的条件下虫子们开始繁殖了(其中包括一些行为并不科学的虫子)；草的数目继续增长，虫子的数目也随之大量地增长，而且虫子们的营养状态普遍比较好，屏幕上的虫子呈现出一片红色；虫子的数目剧烈的增长，而草的数目却在下降；草的数目迅速下降，虫子的数目开始保持为一定的数，同时，虫子的营养状态在下降，有的虫子已经因为找不到食物而死亡；草的数目还在下降，虫子的数目也开始下降；草的数目已经少的可怜(濒临灭绝)，这时虫子的数目在剧烈的下降，大量的虫子死亡。屏幕上到处都是因为饥饿而苍白的虫子在无力地爬；虫子的数目已经下降到可数的量，这时草开始增加了；再过一会儿，草的数量继续上升，幸存的虫子开始可以找到可吃的食物了。

这样的周期一个一个地在屏幕上出现，虫子们也随之变得越来越聪明了。

下面我们将一步一步地实现这个程序。在这样的过程中，我们将综合应用前面章节中的内容，进一步学习如何在程序中使用窗口、资源(包括图标、菜单、对话框等)、计时器以及在窗口中作图等基本的技术。

第一步——建立生物圈

你在学习 C＋＋的过程中是否有这样的体会：几乎所有的面向对象的教科书在开头解释什么是面向对象的时候，总喜欢用动物来举例子。见的最多的是用“狗”来说明什么是类、什么是对象、什么是封装以及什么是继承，这种描述方法非常亲切，我们很容易就弄明白了面向对象的特点(当然具体编写程序是另外一回事)。但是，正当我们沉醉于面向对象的容易理解或自己的理解能力的时候，教科书中的笔锋一转，就开始讲怎样用面向对象语言编写比较抽象的类(比如，文件观察器类)，之后我们再也看不到类似“狗”这样的通俗易懂的类了。

其实，在开始学习面向对象的时候，我们完全可以编写一些类似“狗”这样的类，其中封装了狗的姓名、年龄、家庭住址以及家长作为这个类的数据成员，封装狗的“走”、“跑”、“吃”、“叫”等行为作为狗的成员函数，当然，狗的“生”和“死”就是狗类的构造函数和析构函数了。比如，我们以这个狗类构造出一个狗的对象，设置它的姓名＝“来福”，年龄＝3 等等，然后我们就可以让它在屏幕上跑来跑去了。

狗是一个复杂的动物，想要在屏幕上画出一条狗都很难，更不要说让它跑来跑去的了。因此，我们找到了一种合适的动物，就是“虫子”，因为虫子在屏幕中可以简单到只是一个点。事实上，在下面的程序中，我们的虫子是由五个点组成的，它真的很象一条长长的虫子。在我们的程序中，虫子是一种食草动物，它们用长长的身体到处爬行，一旦碰到了草就将它吞下，它们身体的颜色是可以改变的，从白色到深红色，这取决与它们体内的能量的多少，能量越多颜色越深。我们很容易看出某个虫子的状态如何，因此，我们在观

察这个“生物圈”的时候经常会说:“看!这个虫子要死了!……那个虫子要生了!”。另外,程序中虫子所在的平面空间是无限的,虫子可以爬到窗口的边界以外,但实际上它又从窗口的另外一边爬了进来。

虫子的类型 TWorm 中封装了虫子的年龄(Age),它表示虫子从出生起已经走了多少步、虫子的当前行动方向(Direction)、虫子身体的当前位置(Point)、虫子的基因(Gene)以及虫子所具有的能量(Blood)。虫子还具有一些行为:虫子可以运动(Run);在运动中,虫子可以向一个方向移动(Move);当碰到食物时虫子可以吃它(Eat)。

下面是虫子类的说明:

```
class TWorm
{
private:
  int    Age;
  int    Direction;
  void   Move();
  void   Eat();
  SwqWindow * Window;
public:
  enum TResult{Nothing=1,Dead=2,Birth=3};
  TWormGene Gene;
  TPoint Point[5];
  intBlood;
  TWorm * NextWorm;
  TResult Run();
  TWorm (SwqWindow * Window, TWormGene ParentGene, int ParentBlood, TPoint APoint);
  virtual ~TWorm ();
};    //{{TWorm}}
```

程序开始时第一代虫子的基因是随机确定的:

```
ThisGene.LeftProb=1+random(50);
ThisGene.RightProb=1+random(50);
```

第二代虫子和以后的虫子将继承第一代虫子的基因:

```
Gene = ParentGene;
```

并在生殖分裂时两个虫子各继承母虫能量的一半:

```
Blood = ParentBlood/2;
```

虫子的基因在继承中还有一定程度的变异,变异量的大小是随机确定的:

```
dl=5-random(11);
dr=5-random(11);
Gene.LeftProb=Gene.LeftProb + dl;
Gene.RightProb=Gene.RightProb + dr;
if (Gene.LeftProb<=0) Gene.LeftProb=0;
if (Gene.RightProb<=0) Gene.RightProb=0;
```

```
if ((Gene.LeftProb+Gene.RightProb)>=100)
{
  Gene.LeftProb=Gene.LeftProb-dl;
  Gene.RightProb=Gene.RightProb-dr;
}
```

下面是虫子类的构造函数，它将根据一定的条件(祖先的基因、位置和能量)构造出一个新的虫子个体：

```
TWorm::TWorm (SwqWindow * Window, TWormGene ParentGene, int ParentBlood, TPoint APoint)
{
}
```

虫子的移动方向可以是四个方向(上下左右)，但它的拐弯方向是两个(左拐弯和右拐弯)。虫子的移动方向完全由基因控制的随机数与当前方向的组合决定。虫子走到了窗口边框的边缘时，它也将毫不在乎地继续往前走，因为它可以从窗口的另一边爬出来。每走一步，虫子的能量就减少 1，而年龄就增加 1。Move 函数完成虫子的移动：

```
void TWorm ::Move()
{

  for(int i=4;i>=1;i--)
  {
      Point[i]=Point[i-1];
  }
  int d;
  d=random(100);
  if ((0<=d)&&(d<=Gene.LeftProb))
  {
      switch (Direction)
      {
      case 1 :
          Point[0].x=Point[0].x-1;
          break;
      case 2 :
          Point[0].y=Point[0].y-1;
          break;
      case 3 :
          Point[0].x=Point[0].x+1;
          break;
      case 4 :
          Point[0].y=Point[0].y+1;
      }
      Direction=Direction-1;
```

```
  }
  ...
  if (Direction>4) Direction = Direction-4;
  if (Direction<1) Direction = Direction+4;
  if (Point[0].x>MAXX) Point[0].x = Point[0].x-MAXX;
  if (Point[0].y>MAXY) Point[0].y = Point[0].y-MAXY;
  if (Point[0].x<0) Point[0].x = Point[0].x+MAXX;
  if (Point[0].y<0) Point[0].y = Point[0].y+MAXY;
  Blood--;
  Age++;
}
```

如果虫子遇到可以吃的食物时就开始吞食，虫子每吃一个单位的食物就可以增长一定量的能量 AddBlood，我们用 Eat 函数来描述虫子吞食食物的过程：

```
void TWorm ::Eat()
{
  if (fmod(Point[0].x,2)! =0) return;
  if (fmod(Point[0].y,2)! =0) return;
  if (Window ->Grass[(Point[0].x)/2][(Point[0].y)/2]==0)
      return;  Blood=Blood+Window ->AddBlood;
  int inX=(Point[0].x)/2;
  int inY=(Point[0].y)/2;
  Window ->Grass[inX][inY]=0;
  Window ->GrassNumber--;
  return;
}
```

虫子的行动都由 Run 函数控制，每执行一步 Run，虫子就往前走一步，并作出一个“吃”的行为(也可能什么都没吃到)，然后进行判断。如果一个虫子满足一定的条件(能量聚集到一定的量)，就返回一个“繁殖”的信息；如果一个虫子的能量消耗光了，它就得牺牲了，函数返回一个“死亡”的信息。TWorm::TResult 是一个枚举类型，它包括了虫子每行动一次的结果：“繁殖”、“死亡”以及“平安无事”。

```
TWorm::TResult TWorm :: Run()
{
  Move();
  Eat();
  if ((Blood>=1000)&&(Age>=10)) return Birth;
  if (Blood<=0 ) return Dead;
  return Nothing;
}
```

另外，我们在虫子类中可以看到这么一句语句：

TWorm * NextWorm;

它是指向一个虫子的指针，这是什么意思呢？其实，这是链表结构中常用的方式，一个对象中包含了指向下一个对象的指针，这样我们就可以顺序地访问这些对象了。难道我们要构造一个虫子链吗？正是。我们可以通过这个链来有效地管理屏幕上的一大堆虫子。

这个虫子链中包含了第一个虫子，我们叫它"头虫"(HeadWorm);ThisWorm 是对虫子链操作的"当前虫";PreWorm 是当前虫的前一个虫子，它参与虫子链的删除或插入操作。GetAWorm 函数顺序获得虫子链中的一个虫子;AddWorm 函数执行在虫子链中插入一条虫子;而 DelWorm 函数执行在虫子链中删除"当前虫"。虫子链的析构函数~TWormList()释放了虫子们所占的内存。下面就是这个虫子链的定义。

```
class TWormList
{
public:
      TWormList(SwqWindow * _Window);
    ~TWormList();
    SwqWindow * Window;
      TWorm * HeadWorm;
    TWorm * PreWorm, * ThisWorm;
    TWorm * GetAWorm();
    BOOL AddWorm(TWorm * AWorm);
    BOOL DelWorm();
};
```

程序主窗口的客户窗口是一个从 TWindow 派生出来的类 SwqWindow。其中封装了一条虫子链 WormList、草的数组 Grass、客户窗口的设备上下文指针 DC 和其他用于操作的数据和方法：

```
class SwqWindow : public TWindow
{
private:
  TWormList * WormList;
  int     AddGrass;
  void    DrawGrass();
  void    DrawWorm();
public:
      int WormNumber;
      TClientDC * DC;
      int AddBlood;
      uint8 Grass[MAXX/2+1][MAXY/2+1];
      uint32 GrassNumber;
      SwqWindow (TWindow * parent, const char far * title = 0, TModule * module = 0);
      virtual ~SwqWindow ();
public:
      virtual void SetupWindow ();
```

```
        virtual void Paint (TDC& dc, bool erase, TRect& rect);
protected:
        void EvTimer (uint timerId);
DECLARE_RESPONSE_TABLE(SwqWindow);
};      //{{SwqWindow}}
```

窗口中的草是虫子赖以生存的食物，草的增长是按照一定的比例进行的。草的生长位置是随机确定的，但在这个生物圈中有一块地方水草茂盛，那就是伊甸园。在那里，由于两个基因中的一个值偏大而导致经常“打转”的虫子生活得很好。程序运行不久，伊甸园中的虫子数目就逐渐增大，这些虫子是慕名而来的吗？显然不是(程序中没有赋予虫子这种自己找食物的本事)，这是自然选择的结果。

草的增长方式是由下面的程序段控制的：

```
//普通地方的草长的方式
  int dg;
  dg=(int)(SetupTransfer.AddGrass.Position/10);
  for (int i=1;i<=dg;i++)
  {
      int ThisX,ThisY;
      ThisX=1+random(MAXX/2);
      ThisY=1+random(MAXY/2);
      if(Grass[ThisX][ThisY]! =1)
      {
          Grass[ThisX][ThisY]=1;
          GrassNumber++;
          DC->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
      }
  }
//伊甸园中草长的方式
  int ydydg;
  ydydg=(int)(SetupTransfer.AddGrass.Position/20);
  for (i=1;i<=ydydg;i++)
  {
      int ThisX,ThisY;
      ThisX=50+random(30);
      ThisY=50+random(30);
      if(Grass[ThisX][ThisY]! =1)
      {
          Grass[ThisX][ThisY]=1;
          GrassNumber++;
          DC->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
      }
  }
```

虫子们在窗口中的行动是窗口的 DrawWorm 函数管理的，它从虫子链中一个一个地将虫子取出，让虫子行动一次(调用 Run 函数)，将虫子身体的最后一节用窗口背景色抹去，而将虫子移动后产生头一节根据虫子的当前能量用一定亮度的红色绘出。而虫子身体的其他部位的颜色不改变，这样不断循环，虫子看上去就象是在爬。而且当虫子吃到草时，它的身体就会从头红到尾。DrawWorm 函数执行一次，所有的虫子都将行动一次。

```
void SwqWindow::DrawWorm()
{
  TWorm * ThisWorm = WormList ->GetAWorm();
  TWorm::TResult AResult;
  while(ThisWorm)
  {
      //最后一节用窗口背景色抹去
      DC ->SetPixel(ThisWorm ->Point[4],TColor(BackColor));

      AResult=ThisWorm ->Run();
      //绘出虫子的头
      DC ->SetPixel(ThisWorm ->Point[0],TColor(RGB(255,255 - ThisWorm ->Blood/4,
                       255 - ThisWorm ->Blood/4)));

      if(AResult= =TWorm::Dead)
      {
          for(int i=0;i<=4;i++)
            DC ->SetPixel(ThisWorm ->Point[i],TColor(BackColor));
          WormList ->DelWorm();
          if (WormNumber<1)
          {
            KillTimer(TIME_WORM);
            MessageBox("这个世界毁灭了!" ,"信息报告");
          }
      }
      if(AResult= =TWorm::Birth)
      {
          TWorm  * ChildWorm1, * ChildWorm2;
          ChildWorm1=new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,
                               ThisWorm ->Point[0]);
          ChildWorm2=new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,
                               ThisWorm ->Point[0]);
          WormList ->AddWorm(ChildWorm1);
          WormList ->AddWorm(ChildWorm2);
          for(int i=0;i<=4;i++)
            DC ->SetPixel(ThisWorm ->Point[i],TColor(BackColor));
          WormList ->DelWorm();
```

```
        }
        ThisWorm = WormList->GetAWorm();
    }
}
```

图 5-4 中列出了“生物圈三号”中定义的各种类。

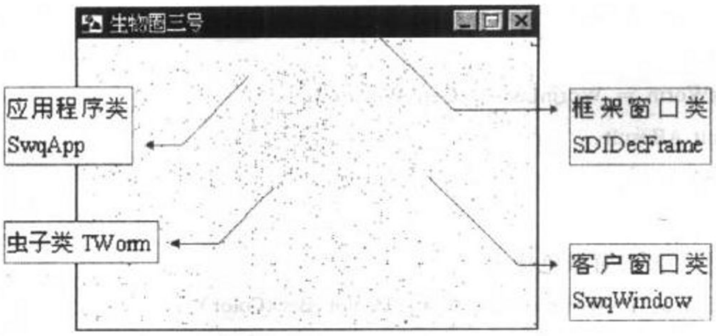


图 5-4　“生物圈三号”程序中的类

在 Windows 中，你最好不要让程序中的一个函数执行太长时间，更不要设计出没完没了的循环，你要知道系统中还有其它程序在运行。那么对于这个程序，问题就是：虫子多长时间走一步，草多长时间长一批？我们可以使用计时器函数来解决这个问题。

SetTimer(TIME_WORM,100);

SetTimer 函数创建一个窗口的计时器，TIME_WORM 是这个计时器的标识，这个计时器每隔 100 ms 就向窗口发出一个计时器消息 WM_TIMER。另外，定义一个 KillTimer 函数来取消窗口中标识为 TIME_WORM 的计时器。一般来说，如果窗口从创建开始就使用计时器，那么 SetTimer 函数就在窗口的 SetupWindow 中调用。

```
void SwqWindow::SetupWindow ()
{
    ...
    SetTimer(TIME_WORM,100);
}
```

撤销计时器应该在窗口的析构函数中调用。

```
SwqWindow::~SwqWindow ()
{
    ...
    KillTimer(TIME_WORM);
}
```

窗口中响应计时器事件的标准函数是 EvTimer 函数，其参数 timerId 用于发出一个消息的计时器标识。

```
void SwqWindow::EvTimer (uint timerId)
{
  TWindow::EvTimer(timerId);
```

```
    DC = new TClientDC( * this) ;
    DC ->SetBkColor(TColor(BackColor));
    DrawGrass();
    DrawWorm();
    delete DC;
    return;
}
```

在 EvTimer 函数中我们创建一个窗口客户区域的设备上下文 DC，然后调用虫子和草的绘制函数，最后我们又将 DC 删除。为什么不在窗口存在的过程中一直保存 DC 对象呢？因为 Windows 系统中设备上下文资源是有限的，如果在程序中一直保存 DC，将会导致系统崩溃。你可以编一个程序，其中申请一个设备上下文(就象申请计时器一样)并在窗口删除的时候再删除它，你会发现这个程序可能运行良好。但是，当你一次一次地运行这个程序(同时运行)，用不了几个回合，你的计算机就死了。

窗口类的成员函数 Paint 是用来响应窗口消息 WM_PAINT，它负责重画客户区域。在这个函数中我们只重画了窗口中的草，而没有画虫子，因为虫子们是运动的，它们会自己爬出来的。

```
void SwqWindow::Paint (TDC& dc, bool erase, TRect& rect)
{
      TWindow::Paint(dc, erase, rect);
for (int X=1;X<=MAXX/2;X++)
  for (int Y=1;Y<=MAXY/2;Y++)
      if (Grass[X][Y]! =0)
          dc.SetPixel(X * 2,Y * 2,TColor::LtGreen);
}
```

图 5-5 所示为 Worm 应用程序的项目文件。

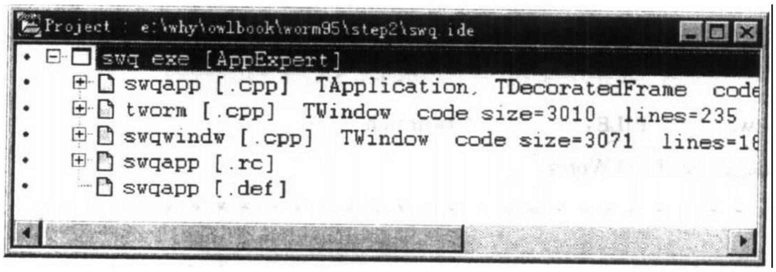


图 5-5　Worm 应用程序的项目文件

```
/*  Project Swq
    FILE:           swqapp.h
    Class definition for SwqApp (TApplication).
****************************************
*/
#if ! defined(__swqapp_h)          // Sentry, use file only if it's not already included.
```

```
#define _swqapp_h
#include <owl\pch.h>
#include "swqapp.rh"              // Definition of all resources.
//
// FrameWindow must be derived to override Paint for Preview and Print.
//
//{{TDecoratedFrame = SDIDecFrame}}
class SDIDecFrame : public TFrameWindow {
public:
    SDIDecFrame (TWindow *parent, const char far *title, TWindow *clientWnd);
    ~SDIDecFrame ();
};    //{{SDIDecFrame}}

//{{TApplication = SwqApp}}
class SwqApp : public TApplication {
private:
public:
    SwqApp ();
    virtual ~SwqApp ();

//{{SwqAppVIRTUAL_BEGIN}}
public:
    virtual void InitMainWindow();
//{{SwqAppVIRTUAL_END}}

//{{SwqAppRSP_TBL_BEGIN}}
protected:
};    //{{SwqApp}}

#endif                                  //_swqapp_h sentry.

/*  Project swq        FILE:        tworm.h
     Class definition for TWorm.
***************************************
*/
#if !defined(_tworm_h)          // Sentry, use file only if it's not already included.
#define _tworm_h
#include <owl\pch.h>
#include "swqwindw.h"

struct TWormGene
{
int LeftProb;
```

```
int RightProb;
};

class TWorm
{
private:
  int      Age;
  int      Direction;
  void     Move();
  void     Eat();
  SwqWindow * Window;
public:
  enum TResult{Nothing=1,Dead=2,Birth=3};
  TWormGene Gene;
  TPoint Point[5];
  intBlood;
  TWorm * NextWorm;
  TResult Run();
  TWorm (SwqWindow * _Window, TWormGene ParentGene, int ParentBlood, TPoint APoint);
  virtual ~TWorm ();
};    //{{TWorm}}

class TWormList
{
public:
    TWormList(SwqWindow * _Window);
    ~TWormList();
    SwqWindow * Window;
      TWorm * HeadWorm;
    TWorm * PreWorm, * ThisWorm;
    TWorm * GetAWorm();
    BOOL AddWorm(TWorm * AWorm);
    BOOL DelWorm();
};
#endif                                   //_tworm_h sentry.

/*   Project Swq
     FILE:          swqwindw.h
     Class definition for SwqWindow (TWindow).
***************************************
*/
#if ! defined(_swqwindw_h)          // Sentry, use file only if it's not already included.
#define_swqwindw_h
```

```
#include <owl\pch.h>
#include "define.h"
//#include "tworm.h"

class TWorm ;
class TWormList;
//{{TWindow = SwqWindow}}
class SwqWindow : public TWindow
{
private:
  TWormList * WormList;
  int      AddGrass;
  void     DrawGrass();
  void     DrawWorm();
public:
    int WormNumber;
    TClientDC * DC;
    int AddBlood;
    uint8 Grass[MAXX/2+1][MAXY/2+1];
    uint32 GrassNumber;
    SwqWindow (TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~SwqWindow ();

//{{SwqWindowVIRTUAL_BEGIN}}
public:
    virtual void SetupWindow ();
    virtual void Paint (TDC& dc, bool erase, TRect& rect);
//{{SwqWindowVIRTUAL_END}}

//{{SwqWindowRSP_TBL_BEGIN}}
protected:
    void EvTimer (uint timerId);
//      void EvPaint ();
//{{SwqWindowRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(SwqWindow);
};      //{{SwqWindow}}
#endif                                   //_swqwindw_h sentry.

/*   Project Swq
     FILE:          swqapp.cpp
     Source file for implementation of SwqApp (TApplication).
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
```

```
#include <owl\pch.h>
#include "swqapp.h"
#include "swqwindw.h"                     // Definition of client class.
#include "swqabtdl.h"                     // Definition of about dialog.
#include "define.h"

//{{SwqApp Implementation}}
////////////////////////
// SwqApp
// =====
//
SwqApp::SwqApp () : TApplication("生物圈三号")
{
}
SwqApp::~SwqApp ()
{
    // INSERT>> Your destructor code here.
}
////////////////////////////////
// SwqApp
// =====
// Application intialization.
//
void SwqApp::InitMainWindow ()
{
    if (nCmdShow != SW_HIDE)
            nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL :
nCmdShow;

    SDIDecFrame * frame = new SDIDecFrame(0, GetName(),0);

    // Override the default window style for the main window.
      frame->Attr.Style |= WS_BORDER | WS_CAPTION | WS_CLIPCHILDREN | WS_MINI-
MIZEBOX | WS_SYSMENU | WS_VISIBLE;
      frame->Attr.Style &= ~(WS_CHILD | WS_MAXIMIZEBOX | WS_THICKFRAME);

    //
    // Assign ICON w/ this application.
    //
    frame->SetIcon(this, IDI_SDIAPPLICATION);

    SetMainWindow(frame);
}
```

```
//{{SDIDecFrame Implementation}}

SDIDecFrame::SDIDecFrame (TWindow * parent, const char far * title, TWindow * clientWnd)
        : TFrameWindow(parent, title, clientWnd == 0 ? new SwqWindow(0, "") : clientWnd, true)
{
        // INSERT>> Your constructor code here.
}

SDIDecFrame::~SDIDecFrame ()
{
}

int OwlMain (int , char * [])
{
        try {
                SwqApp app;
                return app.Run();
        }
        catch (xmsg& x) {
            ::MessageBox(0, x.why().c_str(), "Exception", MB_OK);
        }
        return -1;
}

/*    Project swq
        FILE:             tworm.cpp
        Source file for implementation of TWorm
***************************************
*/
#include <owl\pch.h>
#include <math.h>
#include "tworm.h"
#include "swqwindw.h"
#include "define.h"

//{{TWorm Implementation}}
TWorm::TWorm (SwqWindow *_Window, TWormGene ParentGene, int ParentBlood, TPoint APoint)
{
  int dl,dr;
  Window =_Window;
  Gene = ParentGene;
  dl=5 - random(11);
```

```
    dr=5 - random(11);
    Gene.LeftProb=Gene.LeftProb + dl;
    Gene.RightProb=Gene.RightProb + dr;
    if (Gene.LeftProb<=0) Gene.LeftProb=0;
    if (Gene.RightProb<=0) Gene.RightProb=0;
    if ((Gene.LeftProb+Gene.RightProb)>=100)
    {
        Gene.LeftProb=Gene.LeftProb-dl;
        Gene.RightProb=Gene.RightProb-dr;
    }
    Direction=1+random(4);
    Blood = ParentBlood/2;
    Point[0].x = APoint.x;
    Point[0].y = APoint.y;

        for(int i=1;i<=4;i++)
        {
            Point[i].x=Point[0].x;
            Point[i].y=Point[0].y;
        }
    Age = 0;
    NextWorm=0;
}

TWorm::~TWorm ()
{
}

TWorm::TResult TWorm :: Run()
{
   Move();
   Eat();
   if ((Blood>=1000)&&(Age>=10)) return Birth;
   if (Blood<=0 ) return Dead;
   return Nothing;
}

void TWorm ::Move()
{

  for(int i=4;i>=1;i--)
  {
      Point[i]=Point[i-1];
```

```
}
int d;
d=random(100);
if ((0<=d)&&(d<=Gene.LeftProb))
{
    switch (Direction)
    {
    case 1 :
        Point[0].x=Point[0].x-1;
        break;
    case 2 :
        Point[0].y=Point[0].y-1;
        break;
    case 3 :
        Point[0].x=Point[0].x+1;
        break;
    case 4 :
        Point[0].y=Point[0].y+1;
    }
    Direction=Direction-1;
}

if ((d>Gene.LeftProb)&&(d<=Gene.RightProb+Gene.LeftProb))
{
    switch (Direction)
    {
    case 1 :
        Point[0].x=Point[0].x+1;
        break;
    case 2 :
        Point[0].y=Point[0].y+1;
        break;
    case 3 :
        Point[0].x=Point[0].x-1;
        break;
    case 4 :
        Point[0].y=Point[0].y-1;
    }
    Direction=Direction+1;
}

if (d>Gene.RightProb+Gene.LeftProb)
{
```

```
        switch (Direction)
        {
        case 1 :
            Point[0].y=Point[0].y-1;
            break;
        case 2 :
            Point[0].x=Point[0].x+1;
            break;
        case 3 :
            Point[0].y=Point[0].y+1;
            break;
        case 4 :
            Point[0].x=Point[0].x-1;
        }
    }
    if (Direction>4) Direction = Direction-4;
    if (Direction<1) Direction = Direction+4;
    if (Point[0].x>MAXX) Point[0].x = Point[0].x-MAXX;
    if (Point[0].y>MAXY) Point[0].y = Point[0].y-MAXY;
    if (Point[0].x<0)    Point[0].x = Point[0].x+MAXX;
    if (Point[0].y<0)    Point[0].y = Point[0].y+MAXY;
    Blood--;
    Age++;
}

void TWorm ::Eat()
{
    if (fmod(Point[0].x,2)! =0) return;
    if (fmod(Point[0].y,2)! =0) return;
    if (Window ->Grass[(Point[0].x)/2][(Point[0].y)/2]= =0)
        return;
    Blood=Blood+Window ->AddBlood * 2;
    int inX=(Point[0].x)/2;
    int inY=(Point[0].y)/2;
    Window ->Grass[inX][inY]=0;
    Window ->GrassNumber--;
    return;
}

TWormList :: TWormList(SwqWindow * _Window)
{
    Window =_Window;
    HeadWorm = new TWorm(Window, TWormGene(), 800, TPoint());
```

```
    ThisWorm = HeadWorm;
    for (int i=1;i<=Window->WormNumber;i++)
    {
        TPoint ThisPoint;
        TWormGene ThisGene;
        ThisGene.LeftProb=1+random(50);
        ThisGene.RightProb=1+random(50);
        ThisPoint.x=random(MAXX);
        ThisPoint.y=random(MAXY);

        ThisWorm->NextWorm = new TWorm(Window,ThisGene,800.ThisPoint);
        ThisWorm = ThisWorm->NextWorm;
    }
    ThisWorm->NextWorm=0;
    PreWorm = HeadWorm;
    ThisWorm= PreWorm->NextWorm;
}

TWormList :: ~TWormList()
{
    ThisWorm = HeadWorm->NextWorm;
    while(ThisWorm)
    {
        HeadWorm->NextWorm = ThisWorm->NextWorm;
        delete ThisWorm;
        ThisWorm = HeadWorm->NextWorm;
    }
    delete HeadWorm;
    Window->WormNumber = 0;
}

TWorm * TWormList :: GetAWorm()
{
    if(ThisWorm==0)
    {
        PreWorm = HeadWorm;
        ThisWorm= PreWorm->NextWorm;
    }
    else
    {
        PreWorm = ThisWorm;
        ThisWorm= ThisWorm->NextWorm;
    }
```

```
    return ThisWorm;
  }

  BOOL TWormList :: AddWorm(TWorm * AWorm)
  {
    if(AWorm==0) return false;
    AWorm->NextWorm = HeadWorm->NextWorm;
    HeadWorm->NextWorm = AWorm;
    Window->WormNumber++;
    if(PreWorm->NextWorm! =ThisWorm)
    {
        PreWorm = PreWorm->NextWorm;
    }
    return true;
  }

  BOOL TWormList :: DelWorm()
  {
    if(ThisWorm==0) return false;
    PreWorm->NextWorm = ThisWorm->NextWorm;
    delete ThisWorm;
    ThisWorm = PreWorm->NextWorm;
    Window->WormNumber--;
    return true;
  }

/*   Project Swq          FILE:          swqwindw.cpp
       Source file for implementation of SwqWindow (TWindow).
****************************************
*/
#include <owl\pch.h>
#include <stdio.h>
#include "tworm.h"
#include "swqapp.h"
#include "define.h"
#include "swqwindw.h"

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(SwqWindow, TWindow)
//{{SwqWindowRSP_TBL_BEGIN}}
```

```
        EV_WM_TIMER,
//{{SwqWindowRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{SwqWindow Implementation}}
//////////////////////////////////
// SwqWindow
// =======
// Construction/Destruction handling.
SwqWindow::SwqWindow (TWindow * parent, const char far * title, TModule * module)
        : TWindow(parent, title, module)
{
   //window
   Attr.W = MAXX;
   Attr.H = MAXY;
   SetBkgndColor(BackColor);
   //worm
   AddBlood = 50;
   AddGrass = 30;
   WormNumber=20+random(20);
   WormList = new TWormList(this);
   //gress
   for (int i=1; i<MAXX/2; i++)
   for (int j=1;j<=MAXY/2; j++)
        Grass[i][j]=0;
   GrassNumber=300+random(250);
   for (i=1;i<=GrassNumber;i++)
   {
            TPoint NewPoint;
            NewPoint.x=random(MAXX/2);
            NewPoint.y=random(MAXY/2);
            if (Grass[NewPoint.x][NewPoint.y]==1) i--;
            Grass[NewPoint.x][NewPoint.y]=1;
   }
}

SwqWindow::~SwqWindow ()
{
        Destroy();
        KillTimer(TIME_WORM);
        delete WormList;
}
void SwqWindow::EvTimer (uint timerId)
```

```
{
  TWindow::EvTimer(timerId);
  DC = new TClientDC( * this) ;
  DC ->SetBkColor(TColor(BackColor));
  DrawGrass();
  DrawWorm();
  delete DC;
  return;
}

void SwqWindow::SetupWindow ()
{
      TWindow::SetupWindow();
      SetTimer(TIME_WORM,100);
}

void SwqWindow::DrawGrass()
{
  int dg;
  dg=(int)(AddGrass/10);
  for (int i=1;i<=dg;i++)
  {
      int ThisX,ThisY;
      ThisX=1+random(MAXX/2);
      ThisY=1+random(MAXY/2);
      if(Grass[ThisX][ThisY]! =1)
      {
          Grass[ThisX][ThisY]=1;
          GrassNumber++;
          DC ->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
      }
  }
  int ydydg;
  ydydg=(int)(AddGrass/20);
  for (i=1;i<=ydydg;i++)
  {
      int ThisX,ThisY;
      ThisX=50+random(30);
      ThisY=50+random(30);
      if(Grass[ThisX][ThisY]! =1)
      {
          Grass[ThisX][ThisY]=1;
          GrassNumber++;
```

```
            DC ->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
        }
    }
}

void SwqWindow::DrawWorm()
{
  TWorm * ThisWorm = WormList ->GetAWorm();
  TWorm::TResult AResult;
  while(ThisWorm)
  {
      //HideAWorm(ThisWorm);
      DC ->SetPixel(ThisWorm ->Point[4],TColor(BackColor));

      AResult=ThisWorm ->Run();
      //ShowAWorm(ThisWorm);
      DC ->SetPixel(ThisWorm ->Point[0],TColor(RGB(255,255 - ThisWorm ->Blood/4,255 - This-
Worm ->Blood/4)));

      if(AResult= =TWorm::Dead)
      {
          for(int i=0;i<=4;i++)
            DC ->SetPixel(ThisWorm ->Point[i],TColor(BackColor));
          WormList ->DelWorm();
          if (WormNumber<1)
          {
            KillTimer(TIME_WORM);
            MessageBox("这个世界毁灭了!" ,"信息报告");
          }
      }
      if(AResult= =TWorm::Birth)
      {
          TWorm * ChildWorm1, * ChildWorm2;
          ChildWorm1=new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,
                                ThisWorm ->Point[0]);
          ChildWorm2=new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,
                                ThisWorm ->Point[0]);
          WormList ->AddWorm(ChildWorm1);
          WormList ->AddWorm(ChildWorm2);
          for(int i=0;i<=4;i++)
            DC ->SetPixel(ThisWorm ->Point[i],TColor(BackColor));
          WormList ->DelWorm();
      }
```

```
            ThisWorm = WormList->GetAWorm();
        }
    }

    void SwqWindow::Paint (TDC& dc, bool erase, TRect& rect)
    {
        TWindow::Paint(dc, erase, rect);

        // INSERT>> Your code here.
      for (int X=1;X<=MAXX/2;X++)
      for (int Y=1;Y<=MAXY/2;Y++)
          if (Grass[X][Y]! =0)
              dc.SetPixel(X * 2,Y * 2,TColor::LtGreen);
    }
```

第二步——显示状态

为了方便对生物圈的观察，我们在窗口中加入状态条(见图5-6)。为此，我们给客户窗口类增加一个绘制状态条的函数 DrawMessageBar。在窗口中绘图要使用设备上下文，OWL为此提供了一种方便的途径，就是使用设备上下文类 TDC。DrawMessageBar 函数中使用了一系列 TDC 类的成员函数：

```
dc.FillRect(0,MAXY+1,MAXX,MAXY+ADDY, TBrush(TColor::LtGray));
dc.SelectObject(TPen(TColor::White));
dc.MoveTo(0,MAXY+1);
dc.LineTo(MAXX,MAXY+1);
dc.TextOut(10,MAXY+5,"虫子的数量:");
```

其中，FillRect 函数在指定的区域中填充指定的颜色。程序中在窗口的底部填充一个灰色条状区域，作为状态条的基础。

SelectObject 函数让 DC 选择一个对象(可以是画笔、刷子、字体或者调色板)用来绘图，参数 TPen 是画笔类，TColor 是颜色类，而 White 是颜色类的一个静态数据成员。

MoveTo 函数将画笔的当前位置移动到指定的位置。

LineTo 函数用当前画笔从当前位置到指定位置画一条线段。为了使状态条看起来立体一些，程序使用画线函数在状态条的边缘绘制四条明暗不同的边框。

TextOut 函数在状态条中用当前画笔绘制出特定的文字(比如，“虫子的数量:”)。但是，这个函数只能绘制字符串，如果我们需要绘制数字或其他类型数据，就需要将它们转换成字符串。sprintf 函数在将其他类型转换为字符串时非常有效。程序使用文本输出绘制了生物的状态。

```
void SwqWindow::DrawMessageBar(TDC& dc)
{
    dc.FillRect(0,MAXY+1,MAXX,MAXY+ADDY, TBrush(TColor::LtGray));
    dc.SelectObject(TPen(TColor::White));
    dc.MoveTo(0,MAXY+1);
```

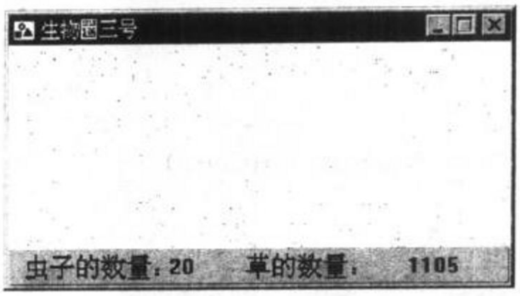


图 5-6　在窗口中绘制状态条

```
    dc.LineTo(MAXX,MAXY+1);
    dc.MoveTo(0,MAXY+1);
    dc.LineTo(0,MAXY+ADDY);
    dc.SelectObject(TPen(TColor::Gray));
    dc.MoveTo(MAXX-1,MAXY+ADDY-1);
    dc.LineTo(MAXX-1,MAXY+2);
    dc.MoveTo(MAXX-1,MAXY+ADDY-1);
    dc.LineTo(0,MAXY+ADDY-1);
    TColor OldBKColor=dc.SetBkColor(TColor::LtGray);
    dc.SelectObject(TPen(TColor::Black));
    dc.TextOut(10,MAXY+5,"虫子的数量:");
    dc.TextOut(210,MAXY+5,"草的数量:");
    char buffer[10];
    sprintf(buffer," %d", WormNumber);
    dc.TextOut(100,MAXY+5,buffer);
    printf(buffer," %d",GrassNumber);
    dc.TextOut(310,MAXY+5,buffer);
    dc.SetBkColor(OldBKColor);
}
```

相应地，在客户窗口类中增加 DrawMessageBar 函数：

```
class SwqWindow : public TWindow
{
private:
  TWormList * WormList;
  int     AddGrass;
  void    DrawGrass();
  void    DrawWorm();
  void    DrawMessageBar(TDC& dc);
  ......
  DECLARE_RESPONSE_TABLE(SwqWindow);
};      //{{SwqWindow}}
```

在 Paint 函数中增加绘制状态条的语句：

```
void SwqWindow::Paint (TDC& dc, bool erase, TRect& rect)
{
  TWindow::Paint(dc, erase, rect);
  for (int X=1;X<=MAXX/2;X++)
  for (int Y=1;Y<=MAXY/2;Y++)
      if (Grass[X][Y]! =0)
          dc.SetPixel(X * 2,Y * 2,TColor::LtGreen);
  DrawMessageBar(dc);
}
```

在 EvTimer 函数中增加绘制状态条的语句：

```
void SwqWindow::EvTimer (uint timerId)
{
  TWindow::EvTimer(timerId);
  DC = new TClientDC( * this) ;
  DC ->SetBkColor(TColor(BackColor));
  DrawGrass();
  DrawWorm();
  DrawMessageBar( * DC);
  delete DC;
  return;
}
```

事实上，你可能发现前面的方法并不是十分理想。因为，每当程序调用 EvTimer 函数一次，状态条就要被重画一次，如果显示速度不是很快，状态条就会有明显的闪烁。当然我们可以在每次调用 EvTimer 函数的时候只重画改变的部分(改变的数字)，但在 Paint 函数中我们仍然需要重画这个状态条。

为了避免用设备上下文来绘制状态条，我们想在窗口中增加了一系列静态控件对象和一个按钮对象来实现同样的任务(见图 5-7)。

```
TButton   * MessageBar;
TStatic   * WormTitle,
          * WormNum,
          * GrassTitle,
          * GrassNum,
          * AddGrassTitle,
          * AddGrassNum,
          * AddBloodTitle,
          * AddBloodNum;
```

使用这些静态控件分别显示虫子的数目、草的数目以及虫子的能量增加率和草的增加率；按钮控件是用来显示状态条的(实际上，一个长长的按钮就很象一个状态条)。静态控件位置在按钮的正上方，因为它们的底色都是浅灰色，所以我们不会看到它们是有一个一个的控件组成的。

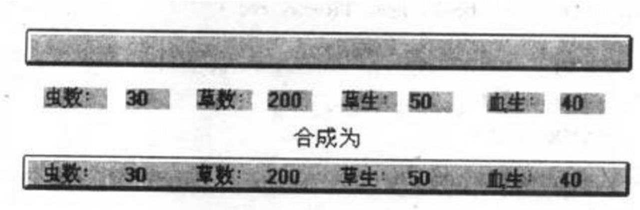


图 5-7 用一系列控件作为状态条的设想

```
MessageBar = new TButton(this, 200, "", 0,MAXY+1,MAXX,MAXY+ADDY);
WormTitle = new TStatic(this, TS_WT, "虫数:",20,MAXY+5,50,18,6);
WormNum = new TStatic(this, TS_WN, "", 70,MAXY+5,50,18,6);
GrassTitle = new TStatic(this, TS_GT, "草数:", 120, MAXY+5, 50,18,6);
GrassNum = new TStatic(this, TS_GN, "", 170,MAXY+5,50,18,6);
AddGrassTitle = new TStatic(this, TS_AGT, "草生:", 230, MAXY+5, 50 ,18,6);
AddGrassNum = new TStatic(this, TS_AGN, "", 280, MAXY+5,50,18,6);
AddBloodTitle = new TStatic(this, TS_ABT, "血生:",310, MAXY+5, 50 ,18,6);
AddBloodNum = new TStatic(this, TS_ABN, "", 360, MAXY+5,50,18,6);

MessageBar ->Create();
WormTitle ->Create();
WormNum ->Create();
GrassTitle ->Create();
GrassNum ->Create();
AddGrassTitle ->Create();
AddGrassNum ->Create();
AddBloodTitle ->Create();
AddBloodNum ->Create();
```

即使这样，还有问题。当用户在窗口中点击按钮时，按钮就显示在静态控件的上面了，也就是说，我们看不到要显示的信息。如何让按钮对用户的操作置之不理呢？很简单，我们只要将按钮的状态设置为灰色(不响应如何操作)：

```
MessageBar ->EnableWindow(false);
```

这样状态条的设计就完成了(见图 5-8)。

其实，我们还可以用一个比较大的静态控件取代作为状态条基础的按钮控件，这样做唯一的缺点就是状态条没有立体感。

既然是用静态控件来显示状态，就需要对静态控件进行操作，比如：

```
char buffer[10];
sprintf(buffer," %d", GrassNumber);
GrassNum ->SetText(buffer);
```

相应地，Paint 函数变成如下代码：

```
void SwqWindow::Paint (TDC& dc, bool erase, TRect& rect)
{
  TWindow::Paint(dc, erase, rect);
```

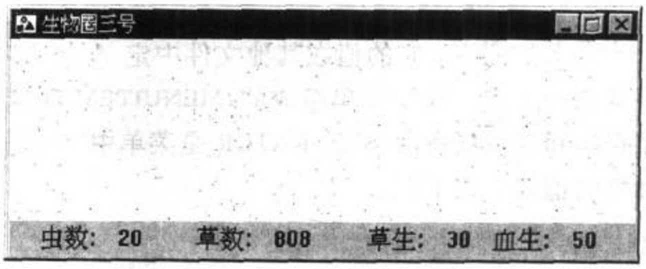


图 5-8 加入状态条后的生物圈

```
    for (int X=1;X<=MAXX/2;X++)
    for (int Y=1;Y<=MAXY/2;Y++)
        if (Grass[X][Y]! =0)
            dc.SetPixel(X * 2,Y * 2,TColor::LtGreen);
    char buffer[10];
    sprintf(buffer," %d", WormNumber);
    WormNum ->SetText(buffer);
    sprintf(buffer," %d", GrassNumber);
    GrassNum ->SetText(buffer);
    sprintf(buffer," %d", AddGrass);
    AddGrassNum ->SetText(buffer);
    sprintf(buffer," %d", AddBlood);
    AddBloodNum ->SetText(buffer);
}
```

第三步——加入菜单条和对话框

一个比较完整的 Windows 程序应该具有用户可以控制的部件，菜单条和对话框都有可以接收用户命令和信息的标准方式。

在资源文件中可视化地添加菜单是比较容易的，我们也可以用文本编辑菜单资源，将下面的代码加入资源文件中：

```
SDI_MENU MENU
{
  POPUP "游戏"
  {
    MENUITEM "设置", CM_SETUP
    MENUITEM "开始", CM_BEGIN
    MENUITEM "暂停", CM_STOP
    MENUITEM SEPARATOR
    MENUITEM "退出\tAlt+F4", CM_EXIT
  }
  POPUP "帮助"
  {
    MENUITEM "关于...", CM_HELPABOUT
  }
```

```
}
```

SDI_MENU 是一个菜单的名字，它的值在其他文件中定义；POPUP 项定义一个弹出菜单，程序中的一个菜单条是由多个弹出菜单组成的；MENUITEM 项定义了一个菜单项，后面有菜单项显示的字符串和菜单项命令；SEPARATOR 是菜单中的一条分割线。

菜单和菜单项命令的值定义如下：

```
#define SDI_MENU            100
#define CM_STOP             24337
#define CM_BEGIN            24336
#define CM_SETUP            24335
#define CM_EXIT             24310
#define CM_HELPABOUT        2009
```

如果是给应用程序的主窗口加入菜单条，一般在应用程序类的 InitMainWindow 函数中实现它，使用窗口类的 AssignMenu 函数完成这一功能：

```
void SwqApp::InitMainWindow ()
{
        ...
    SDIDecFrame *frame = new SDIDecFrame(0, GetName(),0);
        ...
    frame->AssignMenu(SDI_MENU);
        ...
    SetMainWindow(frame);
}
```

如果使用 AppExpert，那么当你在资源文件中增加了菜单条后，Events 窗口就会增加菜单项命令树。当你在某一项上按下设备右键，就会看到一个弹出的对话框，要你输入处理这个命令的函数。注意：这时增加的处理函数就成为当前类的成员函数，而当前对象是当前 Classes 窗口中所指的类。比如，我们在 Classes 窗口中选中 SwqWindow 类，并且在 Events 窗口中增加“关于”命令的处理函数程序中 SwqWindow 类会增加了一个成员函数：

```
class SwqWindow : public TWindow
{
        ...
    void CmAbout();
        ...
DECLARE_RESPONSE_TABLE(SwqWindow);
}; //{{SwqWindow}}
```

在事件响应表中增加了一条语句：

```
DEFINE_RESPONSE_TABLE1(SwqWindow, TWindow)
//{{SwqWindowRSP_TBL_BEGIN}}
        ...
    EV_COMMAND(CM_HELPABOUT, CmAbout),
```

```
        ...
//{{SwqWindowRSP_TBL_END}}
END_RESPONSE_TABLE;
```

在程序中增加了一个函数体：

```
void SwqWindow::CmAbout()
{
        // INSERT>> Your constructor code here.
}
```

我们很容易地把它变成一个命令处理函数：

```
void SwqWindow::CmAbout()
{
  SwqAboutDlg aDlg(this);
  aDlg.Execute();}
```

当然，所有这一切我们都可以手工完成。

为了方便，我们让 SwqWindow 类响应了几乎全部的菜单项命令(除了"退出"命令是由程序响应的)：

```
DEFINE_RESPONSE_TABLE1(SwqWindow, TWindow)
//{{SwqWindowRSP_TBL_BEGIN}}
          ...
    EV_COMMAND(CM_SETUP, CmSetup),
    EV_COMMAND(CM_HELPABOUT, CmAbout),
    EV_COMMAND(CM_BEGIN, CmBegin),
    EV_COMMAND(CM_STOP, CmStop),
          ...
//{{SwqWindowRSP_TBL_END}}
END_RESPONSE_TABLE;
```

其中 CmBegin 命令是在窗口中设置一个计时器，使得在选择菜单项"开始"之后生物圈才开始启动，而不是象原先那样一启动程序虫子们就开始到处爬：

```
void SwqWindow::CmBegin()
{
      SetTimer(TIME_WORM,100);
}
```

而相应的 CmStop 函数是在窗口中删除这个计时器，这样生物圈就停止运动了：

```
void SwqWindow::CmStop()
{
      KillTimer(TIME_WORM);
}
```

再次选择"开始"菜单项，生物圈会接着运行。

另外，我们不希望用户在选择"开始"菜单项以后再一次选择这个菜单项，也就是说让设置计时器的语句执行两次，或者连续选择执行两"暂停"菜单项命令。为了防治程序连续两设置或删除计时器，我们在程序中使用了一个标志 IsRuning，它是一个 BOOL 类型变量，它标志生物圈是否在运行(计时器是否被设置)。如果 IsRuning 的值为 true，则计时器已经设置，否则计时器没有设置。在前面的有关集成环境的图中，我们可以看到每一个菜单项命令都有两个可选操作:Command 和 Command Enable。关于 Command 你已经比较清楚了，它就是当用户选择这个菜单项时要作出的相应，而 Command Enable 则是判断菜单项是否是可选的。如果菜单项是不可选的，它就会变成灰色，用户选择这样的菜单项时程序将没有如何反应。与 Command 一样，我们可以让集成环境来完成大部分工作。在回答对话框之后，事件响应表中会增加 EV_COMMAND_ENABLE 项目：

```
DEFINE_RESPONSE_TABLE1(SwqWindow, TWindow)
//{{SwqWindowRSP_TBL_BEGIN}}
......
    EV_COMMAND_ENABLE(CM_BEGIN, CmBeginEnable),
    EV_COMMAND_ENABLE(CM_STOP, CmStopEnable ),
//{{SwqWindowRSP_TBL_END}}
END_RESPONSE_TABLE;
```

其中，CM_BEGIN 是菜单项命令的标识，CmBeginEnable 是你在对话框中给这个事件的响应函数起的名字。相应的响应函数为：

```
//
//如果"生物圈三号"已经在运行，关闭菜单项命令'游戏\开始'
//
void SwqWindow::CmBeginEnable(TCommandEnabler& commandHandler)
{
  commandHandler.Enable(! IsRuning);
}

//
// 如果"生物圈三号"没有在运行，关闭'游戏\暂停'菜单项
//
void SwqWindow::CmStopEnable(TCommandEnabler& commandHandler)
{
  commandHandler.Enable(IsRuning);
}
```

如何知道生物圈是否在运行？通过测试 IsRuning 参数的值。所以 IsRuning 的值应该在启动和停止生物圈的时候改变：

```
void SwqWindow::CmBegin()
{
    SetTimer(TIME_WORM,100);
    IsRuning = true;
```

```
}
void SwqWindow::CmStop()
{
        KillTimer(TIME_WORM);
        IsRuning = false;
}
```

程序中有两个对话框，一个是“帮助”对话框，它显示了程序的信息，通过响应菜单项命令“帮助\关于”来执行这个模式对话框。这个对话框是一个简单的对话框(见图5-9)，它没有什么参数需要传输：

```
void SwqWindow::CmAbout()
{
  SwqAboutDlg aDlg(this);
  aDlg.Execute();
}
```

程序中的另外一个对话框是“参数设置”对话框(见图 5-10)，它用两个滚动条来设置程序中的两个参数:生物圈中草的增长率 AddGrass 和虫子能量的增长率 AddBlood。当用户选择菜单项命令“游戏\设置”时退出这个对话框：

图 5-9 “关于”对话框

```
void SwqWindow::CmSetup()
{
  SwqSetupDlg aDlg(this, IDD_SETUP, SetupTransfer);
  aDlg.Execute();
}
```

SwqSetupDlg 对话框类的定义如下：

```
class SwqSetupDlg : public TDialog
{
  public:
      SwqSetupDlg (TWindow * parent, int resId, TSetupDlgTransfer& ts);
      virtual ~SwqSetupDlg ();
  private:
      TScrollBar *        GrassAddScrollBar;
      TScrollBar *        BloodAddScrollBar;
};      //{{SwqSetupDlg}}
```

其中，包括了两个状态条对象。对话框类的构造函数中设置了对话框的传输缓冲区 ts，这个数据缓冲区是主窗口类的数据成员：

```
SwqSetupDlg::SwqSetupDlg (TWindow * parent, int resId, TSetupDlgTransfer& ts)
      :TDialog(parent, resId)
{
```

```
    GrassAddScrollBar = new TScrollBar(this, IDC_ADDGRASS);
    BloodAddScrollBar = new TScrollBar(this, IDC_ADDBLOOD);
    SetTransferBuffer(&ts);
}
```

对话框数据传输缓冲区结构的定义如下：

```
struct TSetupDlgTransfer
{
    TSetupDlgTransfer();
    TScrollBarData AddGrass;
    TScrollBarData AddBlood;
};
```

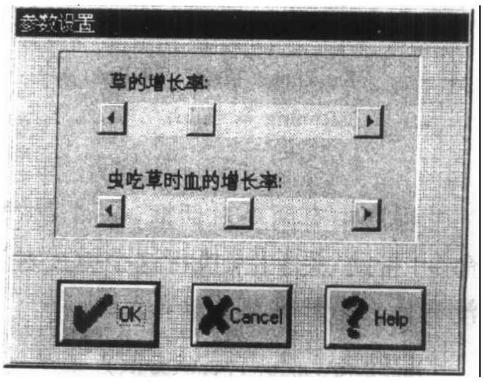


图 5-10　“参数设置”对话框

加入了菜单条和对话框后的生物圈三号如图 5-11 所示，新的 Worm 程序项目文件如图 5-12 所示。

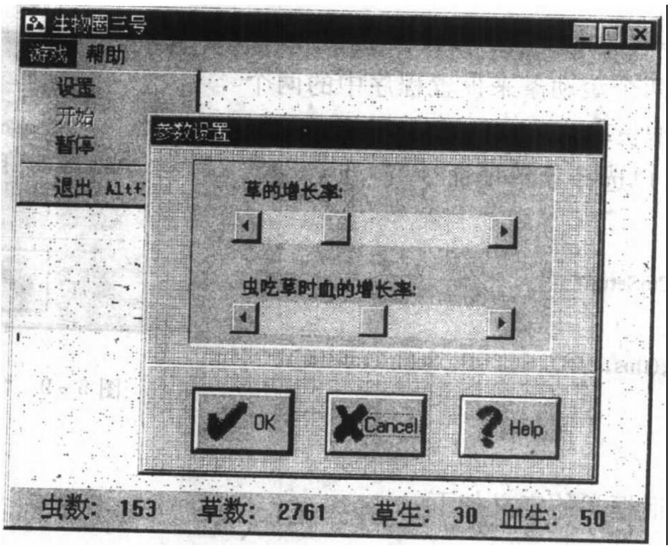


图 5-11　加入菜单条和对话框后的生物圈

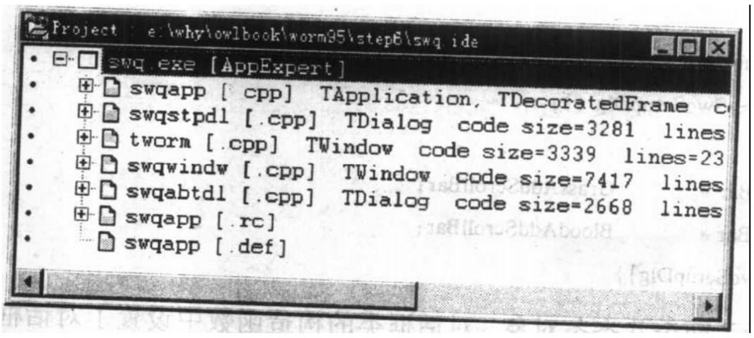


图 5-12　Worm 应用程序项目文件

```
/*  Project Swq
     FILE:            swqapp.h
     Class definition for SwqApp (TApplication).
**************************************
*/
#if ! defined(_swqapp_h)          // Sentry, use file only if it's not already included.
#define_swqapp_h
#include <owl\pch.h>
#include "swqapp.rh"                      // Definition of all resources.

//{{TDecoratedFrame = SDIDecFrame}}
class SDIDecFrame : public TFrameWindow {
public:
    SDIDecFrame (TWindow *parent, const char far *title, TWindow *clientWnd);
    ~SDIDecFrame ();
};    //{{SDIDecFrame}}

//{{TApplication = SwqApp}}
class SwqApp : public TApplication {
public:
    SwqApp ();
    virtual ~SwqApp ();

//{{SwqAppVIRTUAL_BEGIN}}
public:
    virtual void InitMainWindow();
//{{SwqAppVIRTUAL_END}}

//{{SwqAppRSP_TBL_BEGIN}}
protected:
};    //{{SwqApp}}
#endif                                         //_swqapp_h sentry.

/*  Project swq
     FILE:            swqstpdl.h
     Class definition for SwqSetupDlg (TDialog). ***********************
*****************
*/
#if ! defined(_swqstpdl_h)        // Sentry, use file only if it's not already included.
#define_swqstpdl_h
#include <owl\owlpch.h>
#pragma hdrstop
#include "swqapp.rh"                      // Definition of all resources.
```

```
struct TSetupDlgTransfer
{
  TSetupDlgTransfer();
  TScrollBarData AddGrass;
  TScrollBarData AddBlood;
};

//{{TDialog = SwqSetupDlg}}
class SwqSetupDlg : public TDialog
{
  public:
      SwqSetupDlg (TWindow * parent, int resId, TSetupDlgTransfer& ts);
      virtual ~SwqSetupDlg ();
  private:
      TScrollBar *        GrassAddScrollBar;
      TScrollBar *        BloodAddScrollBar;
};    //{{SwqSetupDlg}}
#endif                                          //_swqstpdl_h sentry.

/*   Project swq
      FILE:           tworm.h
      Class definition for TWorm (TWindow).
****************************************************
*/
#if ! defined(_tworm_h)           // Sentry, use file only if it's not already included.
#define_tworm_h
#include <owl\pch.h>
#include "swqwindw.h"

struct TWormGene
{
int LeftProb;
int RightProb;
};

class TWorm
{
private:
  int     Age;
  int     Direction;
  void    Move();
  void    Eat();
```

```
    SwqWindow * Window;
  public:
    enum TResult{Nothing=1,Dead=2,Birth=3};
    TWormGene Gene;
    TPoint Point[5];
    intBlood;
    TWorm * NextWorm;
    TResult Run();
    TWorm (SwqWindow * _Window, TWormGene ParentGene, int ParentBlood, TPoint APoint);
    virtual ~TWorm ();
  };      //{{TWorm}}

  class TWormList
  {
  public:
        TWormList(SwqWindow * _Window);
      ~TWormList();
      SwqWindow * Window;
        TWorm * HeadWorm;
      TWorm * PreWorm, * ThisWorm;
      TWorm * GetAWorm();
      BOOL AddWorm(TWorm * AWorm);
      BOOL DelWorm();
  };
  #endif                                    //_tworm_h sentry.

  /*   Project Swq
       FILE:          swqwindw.h
       Class definition for SwqWindow (TWindow).
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
  */
  #if ! defined(_swqwindw_h)          // Sentry, use file only if it's not already included.
  #define_swqwindw_h
  #include <owl\owlpch.h>
  #pragma hdrstop
  #include "define.h"
  #include "swqstpdl.h"

  class TWorm ;
  class TWormList;
  //{{TWindow = SwqWindow}}
  class SwqWindow : public TWindow
  {
```

```
private:
  TWormList     * WormList;
  TButton       * MessageBar;
  TStatic       * WormTitle,
                * WormNum,
                * GrassTitle,
                * GrassNum,
                * AddGrassTitle,
                * AddGrassNum,
                * AddBloodTitle,
                * AddBloodNum;
  BOOLIsRuning;
  void DrawGrass();
  void DrawWorm();

public:
      TClientDC * DC;
      uint8 Grass[MAXX/2+1][MAXY/2+1];
      int GrassNumber;
      int WormNumber;
      SwqWindow (TWindow * parent, const char far * title = 0, TModule * module = 0);
      virtual ~SwqWindow ();
      virtual void SetupWindow ();
      TSetupDlgTransfer SetupTransfer;
      void CmBeginEnable(TCommandEnabler& commandHandler);
      void CmStopEnable (TCommandEnabler& commandHandler);
//{{SwqWindowVIRTUAL_BEGIN}}
public:
      virtual void Paint (TDC& dc, bool erase, TRect& rect);
//{{SwqWindowVIRTUAL_END}}

//{{SwqWindowRSP_TBL_BEGIN}}
protected:
      void EvTimer (uint timerId);
      void CmSetup();
      void CmAbout();
      void CmBegin();
      void CmStop();
//{{SwqWindowRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(SwqWindow);
};    //{{SwqWindow}}
#endif                                   //_swqwindw_h sentry.
```

```
/*  Project Swq
     FILE:          swqabtdl.h
     Class definition for SwqAboutDlg (TDialog).
*************************************
*/
#if !defined(_swqabtdl_h)          // Sentry, use file only if it's not already included.
#define_swqabtdl_h

#include <owl\pch.h>
#include "swqapp.rh"         // Definition of all resources.

//{{TDialog = SwqAboutDlg}}
class SwqAboutDlg : public TDialog {
public:
   SwqAboutDlg (TWindow *parent, TResId resId = IDD_ABOUT, TModule *module = 0);
   virtual ~SwqAboutDlg ();

};    //{{SwqAboutDlg}}
#endif                              //_swqabtdl_h sentry.

/*  Project Swq
     FILE:          swqapp.cpp
     Source file for implementation of SwqApp (TApplication).
*************************************
*/
#include <owl\pch.h>
#include "swqapp.h"
#include "swqwindw.h"                  // Definition of client class.
#include "swqabtdl.h"                  // Definition of about dialog.
#include "define.h"

//{{SwqApp Implementation}}
////////////////////////////////
// SwqApp
// =====
//
SwqApp::SwqApp () : TApplication("生物圈三号")
{
}

SwqApp::~SwqApp ()
{
     // INSERT>> Your destructor code here.
```

```
}
//////////////////////////////////////
// SwqApp
// =====
// Application intialization.
//
void SwqApp::InitMainWindow ()
{
        if (nCmdShow != SW_HIDE)
                nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL :
nCmdShow;

        SDIDecFrame *frame = new SDIDecFrame(0, GetName(),0);

        // Override the default window style for the main window.
        frame->Attr.Style |= WS_BORDER | WS_CAPTION | WS_CLIPCHILDREN | WS_MINIMIZE-
BOX | WS_SYSMENU | WS_VISIBLE;
        frame->Attr.Style &= ~(WS_CHILD | WS_MAXIMIZEBOX | WS_THICKFRAME);

        //
        // Assign ICON w/ this application.
        //
        frame->SetIcon(this, IDI_SDIAPPLICATION);

        //
        // Menu associated with window and accelerator table associated with table.
        //
        frame->AssignMenu(SDI_MENU);

        SetMainWindow(frame);

}

//{{SDIDecFrame Implementation}}

SDIDecFrame::SDIDecFrame (TWindow *parent, const char far *title, TWindow *clientWnd)
        : TFrameWindow(parent, title, clientWnd == 0 ? new SwqWindow(0, "") : clientWnd, true)
{
        // INSERT>> Your constructor code here.
}

SDIDecFrame::~SDIDecFrame ()
{
```

```
}

int OwlMain (int , char * [])
{
        try {
                SwqApp app;
            app. EnableBWCC();
                return app. Run();
        }
        catch (xmsg& x) {
                ::MessageBox(0, x. why(). c_str(), "Exception", MB_OK);
        }

        return -1;
}

/*    Project swq
      FILE:              swqstpdl. cpp
      Source file for implementation of SwqSetupDlg (TDialog).
***************************************
*/
#include <owl\pch. h>
#include "swqapp. h"
#include "swqstpdl. h"

//{{SwqSetupDlg Implementation}}

SwqSetupDlg::SwqSetupDlg (TWindow * parent, int resId, TSetupDlgTransfer& ts)
        :TDialog(parent, resId)
{
  GrassAddScrollBar = new TScrollBar(this, IDC_ADDGRASS);
  BloodAddScrollBar = new TScrollBar(this, IDC_ADDBLOOD);
  SetTransferBuffer(&ts);
}

SwqSetupDlg::~SwqSetupDlg ()
{
        Destroy();
}

/*    Project swq
      FILE:              tworm. cpp
      Source file for implementation of TWorm
```

```
**************************************
*/
#include <owl\pch.h>
#include <math.h>
#include "tworm.h"
#include "swqwindw.h"
#include "define.h"

//{{TWorm Implementation}}
TWorm::TWorm (SwqWindow * Window, TWormGene ParentGene, int ParentBlood, TPoint APoint)
{
  int dl,dr;
  Window = Window;
  Gene = ParentGene;
  dl=5-random(11);
  dr=5-random(11);
  Gene.LeftProb=Gene.LeftProb + dl;
  Gene.RightProb=Gene.RightProb + dr;
  if (Gene.LeftProb<=0) Gene.LeftProb=0;
  if (Gene.RightProb<=0) Gene.RightProb=0;
  if ((Gene.LeftProb+Gene.RightProb)>=100)
  {
      Gene.LeftProb=Gene.LeftProb-dl;
      Gene.RightProb=Gene.RightProb-dr;
  }
  Direction=1+random(4);
  Blood = ParentBlood/2;
  Point[0].x = APoint.x;
  Point[0].y = APoint.y;

      for(int i=1;i<=4;i++)
      {
          Point[i].x=Point[0].x;
          Point[i].y=Point[0].y;
      }
  Age = 0;
  NextWorm=0;
}

TWorm::~TWorm ()
{
}
```

```
TWorm::TResult TWorm :: Run()
{
  Move();
  Eat();
  if ((Blood>=1000)&&(Age>=10)) return Birth;
  if (Blood<=0 ) return Dead;
  return Nothing;
}

void TWorm ::Move()
{

  for(int i=4;i>=1;i--)
  {
      Point[i]=Point[i-1];
  }
  int d;
  d=random(100);
  if ((0<=d)&&(d<=Gene.LeftProb))
  {
      switch (Direction)
      {
      case 1 :
          Point[0].x=Point[0].x-1;
          break;
      case 2 :
          Point[0].y=Point[0].y-1;
          break;
      case 3 :
          Point[0].x=Point[0].x+1;
          break;
      case 4 :
          Point[0].y=Point[0].y+1;
      }
      Direction=Direction-1;

}

if ((d>Gene.LeftProb)&&(d<=Gene.RightProb+Gene.LeftProb))
{
    switch (Direction)
    {
    case 1 :
```

```
            Point[0].x=Point[0].x+1;
            break;
        case 2 :
            Point[0].y=Point[0].y+1;
            break;
        case 3 :
            Point[0].x=Point[0].x-1;
            break;
        case 4 :
            Point[0].y=Point[0].y-1;
        }
        Direction=Direction+1;
    }

    if (d>Gene.RightProb+Gene.LeftProb)
    {
        switch (Direction)
        {
        case 1 :
            Point[0].y=Point[0].y-1;
            break;
        case 2 :
            Point[0].x=Point[0].x+1;
            break;
        case 3 :
            Point[0].y=Point[0].y+1;
            break;
        case 4 :
            Point[0].x=Point[0].x-1;
        }
    }
    if (Direction>4) Direction = Direction-4;
    if (Direction<1) Direction = Direction+4;
    if (Point[0].x>MAXX) Point[0].x = Point[0].x-MAXX;
    if (Point[0].y>MAXY) Point[0].y = Point[0].y-MAXY;
    if (Point[0].x<0)    Point[0].x = Point[0].x+MAXX;
    if (Point[0].y<0)    Point[0].y = Point[0].y+MAXY;
    Blood--;
    Age++;
}

void TWorm::Eat()
{
```

```
    if (fmod(Point[0].x,2)! =0) return;
    if (fmod(Point[0].y,2)! =0) return;
    if (Window ->Grass[(Point[0].x)/2][(Point[0].y)/2]==0)
        return;
    Blood=Blood+Window ->SetupTransfer.AddBlood.Position * 2;
    int inX=(Point[0].x)/2;
    int inY=(Point[0].y)/2;
    Window ->Grass[inX][inY]=0;
    Window ->GrassNumber--;
    return;
  }

  TWormList :: TWormList(SwqWindow * _Window)
  {
    Window =_Window;
    HeadWorm = new TWorm(Window, TWormGene(), 800, TPoint());
    ThisWorm = HeadWorm;
    for (int i=1;i<=Window ->WormNumber;i++)
    {
        TPoint ThisPoint;
        TWormGene ThisGene;
        ThisGene.LeftProb=1+random(50);
        ThisGene.RightProb=1+random(50);
        ThisPoint.x=random(MAXX);
        ThisPoint.y=random(MAXY);

        ThisWorm ->NextWorm = new TWorm(Window,ThisGene,800,ThisPoint);
        ThisWorm = ThisWorm ->NextWorm;
    }
   ThisWorm ->NextWorm=0;
   PreWorm = HeadWorm;
   ThisWorm= PreWorm ->NextWorm;
 }

 TWormList :: ~TWormList()
 {
   ThisWorm = HeadWorm ->NextWorm;
   while(ThisWorm)
   {
      HeadWorm ->NextWorm = ThisWorm ->NextWorm;
      delete ThisWorm;
      ThisWorm = HeadWorm ->NextWorm;
   }
```

```
    delete HeadWorm;
    Window ->WormNumber = 0;
  }

  TWorm * TWormList :: GetAWorm()
  {
    if(ThisWorm==0)
    {
        PreWorm = HeadWorm;
        ThisWorm= PreWorm ->NextWorm;
    }
    else
    {
        PreWorm = ThisWorm;
        ThisWorm= ThisWorm ->NextWorm;
    }
    return ThisWorm;
  }

  BOOL TWormList :: AddWorm(TWorm * AWorm)
  {
    if(AWorm==0) return false;
    AWorm ->NextWorm = HeadWorm ->NextWorm;
    HeadWorm ->NextWorm = AWorm;
    Window ->WormNumber++;
    if(PreWorm ->NextWorm! =ThisWorm)
    {
        PreWorm = PreWorm ->NextWorm;
    }
    return true;
  }

  BOOL TWormList :: DelWorm()
  {
    if(ThisWorm==0) return false;
    PreWorm ->NextWorm = ThisWorm ->NextWorm;
    delete ThisWorm;
    ThisWorm = PreWorm ->NextWorm;
    Window ->WormNumber--;
    return true;
  }

/*  Project Swq
```

```
        FILE:           swqwindw.cpp
        Source file for implementation of SwqWindow (TWindow).
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 */
#include <owl\pch.h>
#include <stdio.h>
#include "swqstpdl.h"
#include "swqabtdl.h"
#include "tworm.h"
#include "swqapp.rh"
#include "define.h"
#include "swqwindw.h"
TSetupDlgTransfer :: TSetupDlgTransfer()
{
  AddGrass.LowValue = 0;
  AddGrass.HighValue = 100;
  AddGrass.Position = 50;

  AddBlood.LowValue = 0;
  AddBlood.HighValue = 100;
  AddBlood.Position = 50;
}

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(SwqWindow, TWindow)
//{{SwqWindowRSP_TBL_BEGIN}}
      EV_WM_TIMER,
      EV_COMMAND(CM_SETUP, CmSetup),
      EV_COMMAND(CM_HELPABOUT, CmAbout),
      EV_COMMAND(CM_BEGIN, CmBegin),
      EV_COMMAND(CM_STOP, CmStop),
      EV_COMMAND_ENABLE(CM_BEGIN, CmBeginEnable),
      EV_COMMAND_ENABLE(CM_STOP, CmStopEnable ),
//{{SwqWindowRSP_TBL_END}}
END_RESPONSE_TABLE;

//////////////////////////////////////
// SwqWindow
// ==========
// Construction/Destruction handling.
```

```
SwqWindow::SwqWindow (TWindow * parent, const char far * title, TModule * module)
      : TWindow(parent, title, module)
{
          //window
          Attr.W = MAXX;
          Attr.H = MAXY+ADDY;
          SetBkgndColor(BackColor);
  //worm
  SetupTransfer.AddBlood.Position = 50;
  SetupTransfer.AddGrass.Position = 30;
  WormNumber=20+random(100);
  WormList = new TWormList(this);

  //gress
  for (int i=1; i<MAXX/2; i++)
  for (int j=1;j<=MAXY/2; j++)
      Grass[i][j]=0;
  GrassNumber=300+random(250);
  for (i=1;i<=GrassNumber;i++)
  {
          TPoint NewPoint;
          NewPoint.x=random(MAXX/2);
          NewPoint.y=random(MAXY/2);
          if (Grass[NewPoint.x][NewPoint.y]==1) i--;
          Grass[NewPoint.x][NewPoint.y]=1;
  }
  IsRuning = false;
}

SwqWindow::~SwqWindow ()
{
      Destroy();
      if(IsRuning==true) KillTimer(TIME_WORM);
      delete WormList ;
}

void SwqWindow::EvTimer (uint timerId)
{
  TWindow::EvTimer(timerId);
  DC = new TClientDC( * this) ;
  DC ->SetBkColor(TColor(BackColor));
  DrawGrass();
```

```
    DrawWorm();
    delete DC;
    return;
  }

  void SwqWindow::SetupWindow ()
  {
        TWindow::SetupWindow();

        MessageBar = new TButton(this, 200, "", 0,MAXY+1,MAXX,MAXY+ADDY);
        WormTitle = new TStatic(this, TS_WT, "",20,MAXY+5,50,18,6);
        WormNum = new TStatic(this, TS_WN, "", 70,MAXY+5,50,18,6);
        GrassTitle = new TStatic(this, TS_GT, "", 120, MAXY+5, 50,18,6);
        GrassNum = new TStatic(this, TS_GN, "", 170,MAXY+5,50,18,6);
        AddGrassTitle = new TStatic(this, TS_AGT, "", 230, MAXY+5, 50 ,18,6);
        AddGrassNum = new TStatic(this, TS_AGN, "", 280, MAXY+5,25,18,3);
        AddBloodTitle = new TStatic(this, TS_ABT, "",310, MAXY+5, 50 ,18,6);
        AddBloodNum = new TStatic(this, TS_ABN, "", 360, MAXY+5,25,18,3);

        MessageBar ->Create();
        MessageBar ->EnableWindow(false);
        WormTitle ->Create();
        WormNum ->Create();
        GrassTitle ->Create();
        GrassNum ->Create();
        AddGrassTitle ->Create();
        AddGrassNum ->Create();
        AddBloodTitle ->Create();
        AddBloodNum ->Create();
}

void SwqWindow::Paint (TDC& dc, bool erase, TRect& rect)
{
        TWindow::Paint(dc, erase, rect);

        // INSERT>> Your code here.
  for (int X=1;X<=MAXX/2;X++)
  for (int Y=1;Y<=MAXY/2;Y++)
       if (Grass[X][Y]! =0)
            dc.SetPixel(X * 2,Y * 2,TColor::LtGreen);
  char buffer[10];

 sprintf(buffer," %d", WormNumber);
```

```
    WormNum ->SetText(buffer);
    sprintf(buffer,"%d", GrassNumber);
    GrassNum ->SetText(buffer);

    sprintf(buffer,"%d", SetupTransfer.AddGrass.Position);
    AddGrassNum ->SetText(buffer);
    sprintf(buffer,"%d", SetupTransfer.AddBlood.Position);
    AddBloodNum ->SetText(buffer);

    WormTitle ->SetText("虫数:");
    GrassTitle ->SetText("草数:");
    AddGrassTitle ->SetText("草生:");
    AddBloodTitle ->SetText("血生:");
}

void SwqWindow::DrawGrass()
{

    int dg;
    dg=(int)(SetupTransfer.AddGrass.Position/10);
    for (int i=1;i<=dg;i++)
    {
        int ThisX,ThisY;
        ThisX=1+random(MAXX/2);
        ThisY=1+random(MAXY/2);
        if(Grass[ThisX][ThisY]!=1)
        {
            Grass[ThisX][ThisY]=1;
            GrassNumber++;
            DC ->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
        }
    }

    int ydydg;
    ydydg=(int)(SetupTransfer.AddGrass.Position/20);
    for (i=1;i<=ydydg;i++)
    {
        int ThisX,ThisY;
        ThisX=50+random(30);
        ThisY=50+random(30);
        if(Grass[ThisX][ThisY]!=1)
        {
            Grass[ThisX][ThisY]=1;
```

```
            GrassNumber++;
            DC->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
        }
    }

    char buffer[10];
    sprintf(buffer," %d", GrassNumber);
    GrassNum->SetText(buffer);
}

void SwqWindow::DrawWorm()
{
    TWorm * ThisWorm = WormList->GetAWorm();
    TWorm::TResult AResult;
    while(ThisWorm)
    {
        //HideAWorm(ThisWorm);
        DC->SetPixel(ThisWorm->Point[4],TColor(BackColor));

        AResult=ThisWorm->Run();
        //ShowAWorm(ThisWorm);
        DC->SetPixel(ThisWorm->Point[0],TColor(RGB(255,255-ThisWorm->Blood/4,255-This-
Worm->Blood/4)));

        if(AResult==TWorm::Dead)
        {
            for(int i=0;i<=4;i++)
              DC->SetPixel(ThisWorm->Point[i],TColor(BackColor));
            WormList->DelWorm();

            char buffer[10];
            sprintf(buffer," %d", WormNumber);
            WormNum->SetText(buffer);

            if (WormNumber<1)
            {
              KillTimer(TIME_WORM);
              MessageBox("这个世界毁灭了!","信息报告", MB_ICONINFORMATION | MB_OK);
            }
        }
        if(AResult==TWorm::Birth)
        {
            TWorm *ChildWorm1, *ChildWorm2;
```

```
            ChildWorm1 = new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,ThisWorm ->Point
[0]);
            ChildWorm2 = new TWorm(this, ThisWorm ->Gene,ThisWorm ->Blood,ThisWorm ->Point
[0]);
            WormList ->AddWorm(ChildWorm1);
            WormList ->AddWorm(ChildWorm2);
            for(int i=0;i<=4;i++)
              DC ->SetPixel(ThisWorm ->Point[i],TColor(BackColor));
            WormList ->DelWorm();

            char buffer[10];
            sprintf(buffer,"%d", WormNumber);
            WormNum ->SetText(buffer);

        }
        ThisWorm = WormList ->GetAWorm();
    }
}

void SwqWindow::CmSetup()
{
  SwqSetupDlg aDlg(this, IDD_SETUP, SetupTransfer);
  aDlg.Execute();
  char buffer[4];
  sprintf(buffer,"%d", SetupTransfer.AddGrass.Position);
  AddGrassNum ->SetText(buffer);
  sprintf(buffer,"%d", SetupTransfer.AddBlood.Position);
  AddBloodNum ->SetText(buffer);
}

void SwqWindow::CmAbout()
{
  SwqAboutDlg aDlg(this);
  aDlg.Execute();
}

void SwqWindow::CmBegin()
{
      SetTimer(TIME_WORM,100);
      IsRuning = true;
}

void SwqWindow::CmStop()
```

```
{
        KillTimer(TIME_WORM);
        IsRuning = false;
}

void SwqWindow::CmBeginEnable(TCommandEnabler& commandHandler)
{
   commandHandler.Enable(! IsRuning);
}

void SwqWindow::CmStopEnable(TCommandEnabler& commandHandler)
{
   commandHandler.Enable(IsRuning);
}

/*   Project Swq
        FILE:              swqabtdl.cpp
        Source file for implementation of SwqAboutDlg (TDialog).
***************************************
*/
#include <owl\pch.h>
#include "swqapp.h"
#include "swqabtdl.h"

//{{SwqAboutDlg Implementation}}
////////////////////////////////////////
// SwqAboutDlg
// ========
// Construction/Destruction handling.
SwqAboutDlg::SwqAboutDlg (TWindow *parent, TResId resId, TModule *module)
        : TDialog(parent, resId, module)
{
}

SwqAboutDlg::~SwqAboutDlg ()
{
        Destroy();
}

/*   Project Swq
        FILE:              swqapp.rc
        All resources defined here.
***************************************
```

```
*/
#if ! defined(WORKSHOP_INVOKED)
#include <windows.h>
#endif
#include "swqapp.rh"

SDI_MENU MENU
{
  POPUP "游戏"
  {
    MENUITEM "设置", CM_SETUP
    MENUITEM "开始", CM_BEGIN
    MENUITEM "暂停", CM_STOP
    MENUITEM SEPARATOR
    MENUITEM "退出\tAlt+F4", CM_EXIT
  }

  POPUP "帮助"
  {
    MENUITEM "关于...", CM_HELPABOUT
  }
}

// Application ICON
IDI_SDIAPPLICATION ICON "applsdi.ico"

// About box.
IDD_ABOUT DIALOG 22, 19, 144, 82
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
CLASS "BorDlg_Gray"
CAPTION "关于生物圈"
FONT 14, "仿宋体"
{
  CTEXT "Version 1.0", -1, 6, 22, 133, 10, SS_CENTER | SS_NOPREFIX | WS_GROUP
  CTEXT "生物圈三号 遗传与变异", -1, 32, 10, 96, 8, SS_NOPREFIX
  CONTROL "OWL 5.0 应用程序实例 1996.10", -1, "STATIC", SS_CENTER | SS_NOPREFIX |
WS_GROUP, 10, 41, 129, 11
  ICON IDI_SDIAPPLICATION, -1, 7, 4, 12, 12
  CONTROL "Button", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TAB-
STOP, 60, 64, 24, 15
  CONTROL "", -1, "BorShade", BSS_HBUMP | BSS_LEFT | WS_CHILD | WS_VISIBLE, -1, 60,
145, 1
}
```

```
IDD_SETUP DIALOG 68, 65, 189, 124
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
CLASS "bordlg"
CAPTION "参数设置"
FONT 8, "MS Sans Serif"
{
  CONTROL "", -1, "BorShade", BSS_HDIP | BSS_LEFT | WS_CHILD | WS_VISIBLE, 0, 83, 189,
3
  CONTROL "", IDOK, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TAB-
STOP, 20, 92, 37, 25
  CONTROL "", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TAB-
STOP, 76, 92, 37, 25
  CONTROL "", IDHELP, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TAB-
STOP, 132, 92, 37, 25
  SCROLLBAR IDC_ADDGRASS, 35, 24, 119, 12, SBS_HORZ
  CONTROL "草的增长率:", -1, "BorStatic", SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,
39, 12, 110, 10
  CONTROL "虫吃草时血的增长率:", -1, "BorStatic", SS_LEFT | WS_CHILD | WS_VISIBLE |
WS_GROUP, 39, 49, 110, 10
  SCROLLBAR IDC_ADDBLOOD, 35, 59, 119, 12, SBS_HORZ
  CONTROL "", -1, "BorShade", BSS_GROUP | BSS_CAPTION | BSS_LEFT | WS_CHILD | WS_
VISIBLE, 18, 4, 152, 72
}
```

5.2 将"生物圈三号"改造成一个屏幕保护程序

屏幕保护程序是一种比较特殊的可执行程序(*.scr),它们一般是在屏幕黑色的背景下显示一些活动的物体,比如,一个写了字的球或是几条鱼什么的。我们的"生物圈三号"正好满足这样的要求,它们是一些虫子在黑色的屏幕上爬,寻找绿色的草。我们知道,屏幕保护程序本身是不需要菜单条或者对话框这样与用户交流的方式,因为用户只要一按动键盘或是鼠标按钮,程序就退出。但是,在 Windows 系统的 Control Panel 程序的 Desktop 项目中,有一个"Setup"按钮,通常的屏幕保护程序都可以通过这个按钮来设置程序的参数,比如,速度、物体的数目等。也就是说,程序中需要有这样一个设置参数的对话框,它在用户按下"Setup"按钮时弹出。

与前面程序结构不同,因为屏幕保护的窗口比较简单,只要一个黑色背景的无边框和标题条的主窗口。我们没有采用框架窗口和客户窗口的结构,而是直接采用一个框架窗口类 TFrameWindow 的派生类 TScrsvWindow 作为主窗口,它的客户区域就完成用来客户窗口的所有任务。

```
class TScrsvWindow : public TFrameWindow
{
```

```
public:
    TScrsvWindow (TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TScrsvWindow ();
    virtual void SetupWindow ();
    virtual void GetWindowClass (WNDCLASS& wndClass);
    virtual char far * GetClassName ();
......
DECLARE_RESPONSE_TABLE(TScrsvWindow);
};     //{{TScrsvWindow}}
```

主窗口类中仍然包括了生物圈三号的全部环境信息：

```
int     AddGrass;
int     AddBlood;
void    DrawGrass();
void    DrawWorm();
int     WormNumber;
TWormList * WormList;
TClientDC * DC;
uint8 * * Grass;
uint32 GrassNumber;
```

其中，Grass 的定义有所不同。由于用做屏幕保护的生物圈是局限在一个固定的窗口中，我们可以采用二维数组的方式静态地分配内存空间给 Grass 而不会使这个数据段超出 64K 的限制；不过，当我们需要在整个屏幕上显示 Grass 时，数据段的大小可能超限，一种办法是采取其他方式表示 Grass 的情况，另外一种方式是采用动态分配内存的方式。我们选择后者，因为我们希望大量地重用原来的代码而不希望作大量的改动，而且我们也不能确定屏幕的规格大小(可能是 640×480，或者是 800×600)，故不可能向前面的程序那样定义不变的 MAXX 和 MAXY 来描述窗口的大小。在这个程序中，MAXX 和 MAXY 的值显然是程序运行以后才能确定的。Grass 是一个指向指针的指针，这样描述比较难于理解，或者我们可以说它是一个数组的数组。我们给 Grass 分配了内存以后就可以象通常的二维数组那样使用它了。

```
Grass = new uint8 * [MAXX/2+1];
for (int i = 0; i <= MAXX/2; i++)  Grass[i] = new uint8[MAXY/2+1];
for(i=0; i<=MAXX/2; i++)
    for(int j=0; j<=MAXY/2; j++)
        Grass[i][j] = 0;
```

当然，由于是动态分配 Grass 的内存，所以我们需要在窗口的析构函数中释放 Grass 所占的内存空间：

```
for (int i = 0; i <= MAXX/2; i++)
            delete[] Grass[i];
delete[] Grass;
```

对于MAXX和MAXY，我们已经说明了它们不再是程序定义的宏，而是全局变量，它们的取值是由程序当前的运行环境决定的。

```
int MAXX;
int MAXY;
......
TRect r;
::GetWindowRect(GetDesktopWindow(), &r);
MoveWindow(r, TRUE);
MAXX = r.Width();
MAXY = r.Height();
```

主窗口的构造函数中说明了窗口的风格(弹出窗口)和控件风格(总在最上面)，设置了窗口的背景颜色为黑色，并且隐藏了鼠标的光标。

```
TScrsvWindow::TScrsvWindow (TWindow * parent, const char far * title, TModule * module)
      : TFrameWindow(parent, title, 0, false, module)
{
      // INSERT>> Your constructor code here.
  ShowCursor(FALSE);
  Attr.Style = WS_POPUP;
  Attr.ExStyle = WS_EX_TOPMOST;
  SetBkgndColor(BackColor);
}
```

主窗口的SetupWindow函数作了大部分初始化工作，包括获得窗口的客户区域大小，获得屏幕保护程序的设置(保存在control.ini文件中)，构造一系列虫子和草，设置一个计时器，构造一个设备上下文类的对象。

```
void TScrsvWindow::SetupWindow ()
{
  TWindow::SetupWindow();
  GetCursorPos(PrevPt);
  TRect r;
  ::GetWindowRect(GetDesktopWindow(), &r);
  MoveWindow(r, TRUE);
  MAXX = r.Width();
  MAXY = r.Height();
  char settings[4];
::GetPrivateProfileString(ScrnSectionStr, SetGrassStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &AddGrass);
::GetPrivateProfileString(ScrnSectionStr, SetBloodStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &AddBlood);
  WormNumber=20+random(20);
  WormList = new TWormList(this);
  //gress
```

```
    Grass = new uint8 * [MAXX/2+1];          // STEP 1: SET UP THE ROWS.
    for (int i = 0; i <= MAXX/2; i++)
        Grass[i] = new uint8[MAXY/2+1]; // STEP 2: SET UP THE COLUMNS
    for(i=0; i<=MAXX/2; i++)
    for(int j=0; j<=MAXY/2; j++)
        Grass[i][j] = 0;
    GrassNumber=0;
    int GN = 300+random(250);
    for (i=1;i<=GN;i++)
    {
            TPoint NewPoint;
            NewPoint.x=random(MAXX/2);
            NewPoint.y=random(MAXY/2);
            if (Grass[NewPoint.x][NewPoint.y]==0)
            {
              GrassNumber++;
              Grass[NewPoint.x][NewPoint.y]=1;
            }
    }

    SetTimer(TIME_WORM,100);
    DC = new TClientDC(*this) ;
}
```

其中，GetPrivateProfileString 函数是用来在特定的初始化文件中获得信息。它的用法如下：

```
int GetPrivateProfileString
(
LPCSTR lpszSection;             /* 项目 */
LPCSTR lpszEntry;               /* 入口 */
LPCSTR lpszDefault;             /* 缺省字符串 */
LPSTR lpszReturnBuffer;         /* 目标字符串 */
int cbReturnBuffer;             /* 目标字符串长度 */
LPCSTR lpszFilename;            /* 初始化文件名 */
)
```

与其对照的初始化文件中的部分是：

```
[项目]
入口 1=字符串 1
入口 2=字符串 2
```

对于 control.ini 文件中的如下部分：

```
[ScreenSaver.Worm]
AddGrass=70
AddBlood=50
```

我们可以通过 GetPrivateProfileString 函数进行方便的读取，比如我们需要获得 AddGrass 的取值：

```
char settings[4];
GetPrivateProfileString("ScreenSaver. Worm", AddGrass, "50", settings, sizeof(settings), "control. ini");
sscanf(settings, "%d", &AddGrass);
```

如果初始化文件中没有这个项目，目标字符串的值就取缺省字符串("50")。

与前面的程序不同，这个程序的主窗口在初始化的时候就获得了一个设备上下文对象，并一直拥有它(前面的程序总是在用到的时候才申请)。因为屏幕保护程序在运行的时候，其他的程序都不需要显示，所以我们可以通过一直拥有一个客户区域的设备上下文来避免生物圈每运行一步就申请和释放一次 DC 的麻烦。

主窗口的析构函数在窗口关闭以前恢复了鼠标，取消了计时器，删除了设备上下文，清除了草和虫子所占的内存空间。

```
TScrsvWindow::~TScrsvWindow ()
{
  Destroy();
  ShowCursor(TRUE);
  for (int i = 0; i <= MAXX/2; i++)
          delete[] Grass[i];
  delete[] Grass;
  KillTimer(TIME_WORM);
  delete DC;
  delete WormList;
}
```

在屏幕保护程序运行的时候，当用户按下键盘或者移动了鼠标，程序就会自动退出。如何实现这个功能呢？我们需要窗口响应一系列这样的事件，而事件处理函数则是发出一个关闭窗口的消息。响应的事件响应表为：

```
DEFINE_RESPONSE_TABLE1(TScrsvWindow, TFrameWindow)
//{{TScrsvWindowRSP_TBL_BEGIN}}
      EV_WM_ACTIVATE,
      EV_WM_LBUTTONDOWN,
      EV_WM_RBUTTONDOWN,
      EV_WM_ACTIVATEAPP,
      EV_WM_MBUTTONDOWN,
      EV_WM_SYSCOMMAND,
      EV_WM_SYSKEYDOWN,
      EV_WM_MOUSEMOVE,
      EV_WM_KEYDOWN,
//{{TScrsvWindowRSP_TBL_END}}
END_RESPONSE_TABLE;
```

它们都是标准的窗口事件，对应的标准消息处理函数为：

```
void EvActivate (uint active, bool minimized, HWND hWndOther );
void EvLButtonDown (uint modKeys, TPoint& point);
void EvRButtonDown (uint modKeys, TPoint& point);
void EvActivateApp (bool active, HTASK hTask);
void EvMButtonDown (uint modKeys, TPoint& point);
void EvSysCommand (uint cmdType, TPoint& point);
void EvSysKeyDown (uint key, uint repeatCount, uint flags);
void EvMouseMove (uint modKeys, TPoint& point);
void EvKeyDown (uint key, uint repeatCount, uint flags);
```

这些标准消息处理函数的任务通常就是发出一个关闭窗口的消息，比如：

```
void TScrsvWindow::EvLButtonDown (uint, TPoint&)
{
  PostMessage(WM_CLOSE);
}
```

参数设置对话框与前面程序中的基本相同，但由于这个对话框与主窗口之间已经不需要建立直接的联系了(因为它是在 Windows 系统的 Desktop 窗口中运行)，所以它的传输缓冲区就作为对话框中的一个静态数据成员来处理。

```
struct TSetupDlgXfer
{
    TSetupDlgXfer();
    TScrollBarData      AddBlood;
    TScrollBarData      AddGrass;
};
```

与前面的程序不同的是这个缓冲区类型的构造函数：

```
TSetupDlgXfer::TSetupDlgXfer()
{
  char settings[4];

  AddGrass.LowValue = 0;
  AddGrass.HighValue = 100;
::GetPrivateProfileString(ScrnSectionStr, SetGrassStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &(AddGrass.Position));

  AddBlood.LowValue = 0;
  AddBlood.HighValue = 100;
::GetPrivateProfileString(ScrnSectionStr, SetBloodStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &(AddBlood.Position));
}
```

其中，GetPrivateProfileString 函数的作用与前面描述功能相同，它是从 control.ini 文件中获得对话框初始化所需要的参数。对话框类的描述为：

```
class TSetupDlg : public TDialog {
public:
    TSetupDlg (TWindow * parent, TResId resId = ID_SETUP, TModule * module = 0);
    virtual ~TSetupDlg ();
    void CmHelp();
    static TSetupDlgXfer SetupData;
    virtual void CloseWindow (int retVal = 0);
protected:
    TScrollBar * AddBlood;    TScrollBar * AddGrass;
DECLARE_RESPONSE_TABLE(TSetupDlg);
};    //{{TSetupDlg}}
```

其中，CloseWindow 函数是当用户点击 OK 按钮后，由 CmOK 函数调用的，我们希望在对话框关闭之前能将新设置的参数写到 control.ini 文件中：

```
void TSetupDlg::CloseWindow (int retVal)
{
  char settings[4];
  wsprintf(settings, "%d", AddGrass ->GetPosition());
  ::WritePrivateProfileString(ScrnSectionStr, SetGrassStr, settings, "control.ini");
  wsprintf(settings, "%d", AddBlood ->GetPosition());
  ::WritePrivateProfileString(ScrnSectionStr, SetBloodStr, settings, "control.ini");
  TDialog::CloseWindow(retVal);
}
```

WritePrivateProfileString 函数的功能就是将数据写入一个指定的初始化文件中，它的用法与 GetPrivateProfileString 的用法相似：

```
BOOL WritePrivateProfileString
(
LPCSTR lpszSection;      /* 项目 */
LPCSTR lpszEntry;        /* 入口 */
LPCSTR lpszString;       /* 要增加的字符串 */
LPCSTR lpszFilename;     /* 初始化文件名 */
)
```

如果 control.ini 文件中有需要写入的项目，程序就改写它的值，否则程序将在初始化文件中创建这个项目：

```
[ScreenSaver.Worm]
AddGrass=70
AddBlood=50
```

另外，当用户点击"帮助"按钮时将弹出一个有关信息的对话框：

```
void TSetupDlg::CmHelp()
{ (new scrsvAboutDlg(this))->Execute();
}
```

还有一个至关重要的问题是如何让 Desktop 窗口运行参数设置对话框。实际上，Desktop 窗口打开对话框是通过带参数执行程序实现的，如果执行对话框，那么参数为"/c"或"－c"。我们可以检验程序的参数来区分情况，执行程序。程序中，TScrsvApp 类重新定义了 Run 函数：

```
int TScrsvApp::Run ()
{
  if (strcmp(lpCmdLine, "/c") == 0 || strcmp(lpCmdLine, "-c") == 0)
  {
      ConfigureDialog = new TSetupDlg(0, ID_SETUP, this);

      //
      // Get the desktop control panel applet's window handle and make a
      // TWindow out of it.
      //
      HWND deskAppletHWnd = ::FindWindow(WC_DIALOG, "Desktop");
      TWindow * deskAppletWnd = 0;
      if (deskAppletHWnd)
          deskAppletWnd = new TWindow(deskAppletHWnd);

      //
      // Set our config dialog to have the applet window as its parent if
      // available
      //
      if (deskAppletWnd)
          ConfigureDialog ->SetParent(deskAppletWnd);
      ConfigureDialog ->Execute();
      delete ConfigureDialog;

      //
      // When our alias is destructed, it remove its thunk, & restores old proc
      //
      delete deskAppletWnd;
      return Status;
  }
  else return TApplication::Run();
}
```

函数首先判断程序执行时的参数中是否有"/c "或者"－c "，如果没有，程序将执行 TApplication::Run()函数；如果有满足条件的参数，程序将执行参数设置对话框(见图 5-13)。先构造一个对话框对象，然后获得 control panel 窗口的句柄，并用这个句柄构造一个 OWL 窗口对象：

```
HWND deskAppletHWnd = ::FindWindow(WC_DIALOG, "Desktop");
```

```
TWindow * deskAppletWnd = 0;
if (deskAppletHWnd)
  deskAppletWnd = new TWindow(deskAppletHWnd);
```

将这个窗口设置为对话框的父窗口：

```
if (deskAppletWnd)
  ConfigureDialog ->SetParent(deskAppletWnd);
```

执行这个对话框：

```
ConfigureDialog ->Execute();
```

删除对话框和 control panel 窗口的对象：

```
delete ConfigureDialog;
delete deskAppletWnd;
```

然后返回一个状态值。这样对话框就在 control panel 窗口中得到执行。

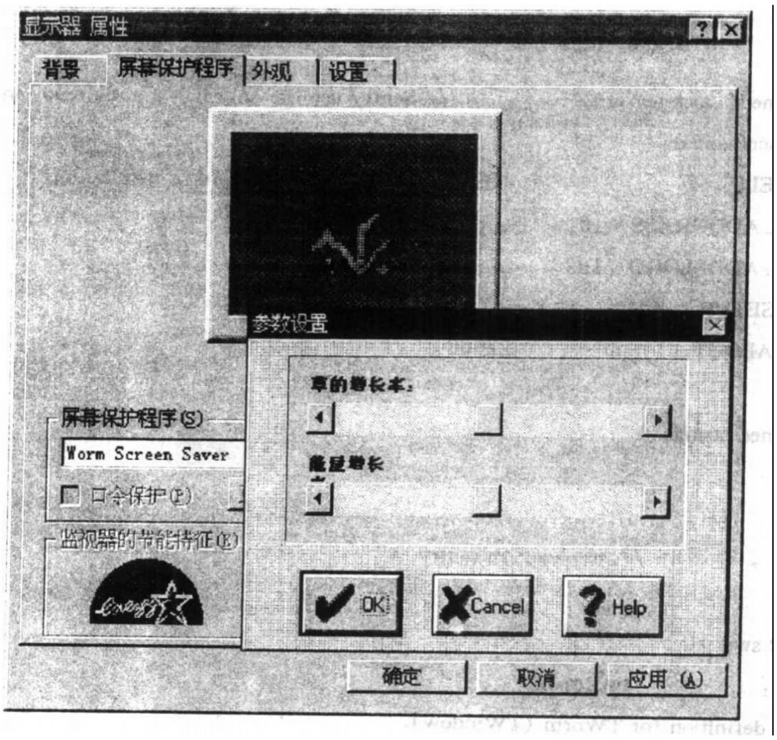


图 5 - 13　在 Screen 窗口中运行参数设置对话框

最后，将程序的目标代码的名字在集成环境的项目窗口中设置为 *.scr，然后将目标程序拷贝到 Windows 所在的路径中。打开 Desktop 窗口，你就可以看到你的屏幕保护程序已经被列入“屏幕保护”一项中。它的名字(比如，“Worm Screen Saver”)是由定义文件(*.def)中的一句确定的：

```
DESCRIPTION 'SCRNSAVE :Worm Screen Saver'
```

图 5 - 14 显示了该屏幕保护程序的项目文件。

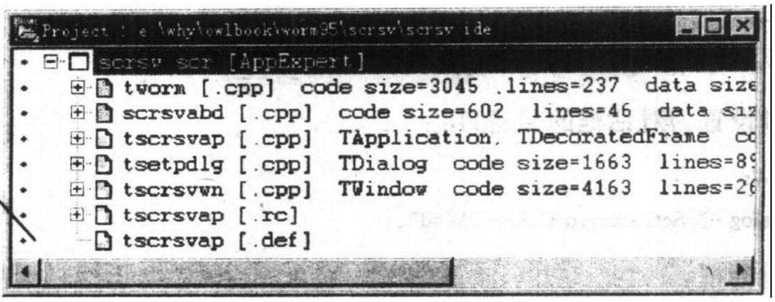


图 5-14 屏幕保护应用程序项目文件

下面是源程序代码：

```
/*  Project scrsv
    FILE:          tscrsvap.h
    Constant definitions for all resources defined in tscrsvap.rc.
**************************************
*/
//#if ! defined(_tscrsvap_rh)          // Sentry use file only if it's not already included.
//#define_tscrsvap_rh
#define IDHELP                 998               // Id of help button
#define IDC_ADDGRASS  101
#define IDC_ADDBLOOD  102
#define ID_SETUP    1
#define ID_ABOUT               22000
//
// OWL defined strings
//
……
//#endif               //_tscrsvap_rh sentry.

/*  Project swq
    FILE:          tworm.h
    Class definition for TWorm (TWindow).
**************************************
*/
#if ! defined(_tworm_h)              // Sentry, use file only if it's not already included.
#define_tworm_h
#include <owl\pch.h>
#include "tscrsvwn.h"
struct TWormGene
{
int LeftProb;
int RightProb;
```

```
};

class TWorm
{
private:
   int      Age;
   int      Direction;
   void     Move();
   void     Eat();
   TScrsvWindow * Window;
public:
   enum TResult{Nothing=1,Dead=2,Birth=3};
   TWormGene Gene;
   TPoint Point[5];
   intBlood;
   TWorm * NextWorm;
   TResult Run();
   TWorm (TScrsvWindow *_Window, TWormGene ParentGene, int ParentBlood, TPoint APoint);
   virtual ~TWorm ();
};    //{{TWorm}}

class TWormList
{
public:
       TWormList(TScrsvWindow *_Window);
     ~TWormList();
     TScrsvWindow * Window;
       TWorm * HeadWorm;
     TWorm * PreWorm, * ThisWorm;
     TWorm * GetAWorm();
     BOOL AddWorm(TWorm * AWorm);
     BOOL DelWorm();
};
#endif                                    //_tworm_h sentry.

/*   Project scrsv
      FILE:          scrsvabd.h
      Class definition for scrsvAboutDlg (TDialog).
**************************************
*/
#if ! defined(_scrsvabd_h)              // Sentry, use file only if it's not already included.
#define_scrsvabd_h
#include <owl\pch.h>
```

```
#include "tscrsvap.rh"                    // Definition of all resources.

//{{TDialog = scrsvAboutDlg}}
class scrsvAboutDlg : public TDialog {
public:
    scrsvAboutDlg (TWindow *parent, TResId resId = ID_ABOUT, TModule *module = 0);
    virtual ~scrsvAboutDlg ();
};    //{{scrsvAboutDlg}}
#endif                                     //_scrsvabd_h sentry.

/*   Project scrsv
      FILE:          tscrsvap.h
      Class definition for TScrsvApp (TApplication).
****************************************
*/
#if !defined(_tscrsvap_h)              // Sentry, use file only if it's not already included.
#define_tscrsvap_h
#include <owl\owlpch.h>
#pragma hdrstop
#include "tscrsvap.rh"                // Definition of all resources.
#include "tsetpdlg.h"

//{{TApplication = TScrsvApp}}
class TScrsvApp : public TApplication {
private:
        TSetupDlg * ConfigureDialog;
public:
        TScrsvApp ();
        virtual ~TScrsvApp ();
public:
        virtual void InitMainWindow();
//{{TScrsvAppVIRTUAL_BEGIN}}
public:
    virtual int Run ();
//{{TScrsvAppVIRTUAL_END}}
};    //{{TScrsvApp}}
#endif                                  //_tscrsvap_h sentry.

/*   Project scrsv
    FILE:          tsetpdlg.h
    Class definition for TSetupDlg (TDialog).
****************************************
*/
```

```
#if ! defined(_tsetpdlg_h)                // Sentry, use file only if it's not already included.
#define_tsetpdlg_h
#include <owl\owlpch.h>
#pragma hdrstop
#include <owl\scrollba.h>
#include "tscrsvap.rh"                // Definition of all resources.

//{{TDialog = TSetupDlg}}
struct TSetupDlgXfer
{
//{{TSetupDlgXFER_DATA}}
    TSetupDlgXfer();
    TScrollBarData     AddBlood;
    TScrollBarData     AddGrass;
//{{TSetupDlgXFER_DATA_END}}
};

class TSetupDlg : public TDialog {
public:
    TSetupDlg (TWindow * parent, TResId resId = ID_SETUP, TModule * module = 0);
    virtual ~TSetupDlg ();
    void CmHelp();
    static TSetupDlgXfer SetupData;
//{{TSetupDlgXFER_DEF}}
protected:
    TScrollBar * AddBlood;
    TScrollBar * AddGrass;
DECLARE_RESPONSE_TABLE(TSetupDlg);
//{{TSetupDlgXFER_DEF_END}}
//{{TSetupDlgVIRTUAL_BEGIN}}
public:
    virtual void CloseWindow (int retVal = 0);
//{{TSetupDlgVIRTUAL_END}}
};    //{{TSetupDlg}}
#endif                                          //_tsetpdlg_h sentry.

/*    Project scrsv
      FILE:             tscrsvwn.h
      Class definition for TScrsvWindow (TWindow).
**************************************
*/
#if ! defined(_tscrsvwn_h)                // Sentry, use file only if it's not already included.
#define_tscrsvwn_h
```

```
#include <owl\owlpch.h>
#pragma hdrstop
#include "tscrsvap.rh"              // Definition of all resources.

class TWorm ;
class TWormList;
//{{TWindow = TScrsvWindow}}
class TScrsvWindow : public TFrameWindow {
public:
    TScrsvWindow (TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TScrsvWindow ();
    virtual void SetupWindow ();
    virtual void GetWindowClass (WNDCLASS& wndClass);
    int     AddGrass;
    int     AddBlood;
    void    DrawGrass();
    void    DrawWorm();
    int     WormNumber;
    TWormList * WormList;
    TClientDC * DC;
    uint8 * * Grass;
    uint32 GrassNumber;
    virtual void Paint (TDC& dc, bool erase, TRect& rect);
//{{TScrsvWindowVIRTUAL_END}}

//{{TScrsvWindowRSP_TBL_BEGIN}}
protected:
    void EvActivate (uint active, bool minimized, HWND hWndOther );
    void EvLButtonDown (uint modKeys, TPoint& point);
    void EvRButtonDown (uint modKeys, TPoint& point);
    void EvActivateApp (bool active, HTASK hTask);
    void EvMButtonDown (uint modKeys, TPoint& point);
    void EvSysCommand (uint cmdType, TPoint& point);
    void EvSysKeyDown (uint key, uint repeatCount, uint flags);
    void EvMouseMove (uint modKeys, TPoint& point);
    void EvKeyDown (uint key, uint repeatCount, uint flags);
    void EvTimer (uint timerId);
//{{TScrsvWindowRSP_TBL_END}}
private:
    TPoint PrevPt;
DECLARE_RESPONSE_TABLE(TScrsvWindow);
};    //{{TScrsvWindow}}
#endif                                  //_tscrsvwn_h sentry.
```

```
/*   Project swq
      FILE:           tworm.cpp
      Source file for implementation of TWorm
***************************************
*/
#include <owl\pch.h>
#include <math.h>
#include "tworm.h"
#include "tscrsvwn.h"
#include "define.h"

extern int MAXX;
extern int MAXY;
//{{TWorm Implementation}}
TWorm::TWorm (TScrsvWindow *_Window, TWormGene ParentGene, int ParentBlood, TPoint APoint)
{
  int dl,dr;
  Window =_Window;
  Gene = ParentGene;
  dl=5-random(11);
  dr=5-random(11);
  Gene.LeftProb=Gene.LeftProb + dl;
  Gene.RightProb=Gene.RightProb + dr;
  if (Gene.LeftProb<=0) Gene.LeftProb=0;
  if (Gene.RightProb<=0) Gene.RightProb=0;
  if ((Gene.LeftProb+Gene.RightProb)>=100)
  {
      Gene.LeftProb=Gene.LeftProb-dl;
      Gene.RightProb=Gene.RightProb-dr;
  }
  Direction=1+random(4);
  Blood = ParentBlood/2;
  Point[0].x = APoint.x;
  Point[0].y = APoint.y;

      for(int i=1;i<=4;i++)
      {
          Point[i].x=Point[0].x;
          Point[i].y=Point[0].y;
      }
Age = 0;
NextWorm=0;
```

```
}

TWorm::~TWorm ()
{
}

TWorm::TResult TWorm :: Run()
{
  Move();
  Eat();
  if ((Blood>=1000)&&(Age>=10)) return Birth;
  if (Blood<=0 ) return Dead;
  return Nothing;
}

void TWorm ::Move()
{
  for(int i=4;i>=1;i--)
  {
      Point[i]=Point[i-1];
  }
  int d;
  d=random(100);
  if ((0<=d)&&(d<=Gene. LeftProb))
  {
      switch (Direction)
      {
      case 1 :
           Point[0]. x=Point[0]. x-1;
           break;
      case 2 :
           Point[0]. y=Point[0]. y-1;
           break;
      case 3 :
           Point[0]. x=Point[0]. x+1;              break;
      case 4 :
           Point[0]. y=Point[0]. y+1;
      }
      Direction=Direction-1;

  }

 if ((d>Gene. LeftProb)&&(d<=Gene. RightProb+Gene. LeftProb))
```

```
{
    switch (Direction)
    {
    case 1 :
        Point[0].x=Point[0].x+1;
        break;
    case 2 :
        Point[0].y=Point[0].y+1;
        break;
    case 3 :
        Point[0].x=Point[0].x-1;
        break;
    case 4 :
        Point[0].y=Point[0].y-1;
    }
    Direction=Direction+1;
}

if (d>Gene.RightProb+Gene.LeftProb)
{
    switch (Direction)
    {
    case 1 :
        Point[0].y=Point[0].y-1;
        break;
    case 2 :
        Point[0].x=Point[0].x+1;
        break;
    case 3 :
        Point[0].y=Point[0].y+1;
        break;
    case 4 :
        Point[0].x=Point[0].x-1;
    }
}
if (Direction>4) Direction = Direction-4;
if (Direction<1) Direction = Direction+4;
if (Point[0].x>MAXX) Point[0].x = Point[0].x-MAXX;
if (Point[0].y>MAXY) Point[0].y = Point[0].y-MAXY;
if (Point[0].x<0)    Point[0].x = Point[0].x+MAXX;
if (Point[0].y<0)    Point[0].y = Point[0].y+MAXY;
Blood--;
Age++;
```

```
}

void TWorm ::Eat()
{
  if (fmod(Point[0].x,2)! =0) return;
  if (fmod(Point[0].y,2)! =0) return;
  if (Window ->Grass[(Point[0].x)/2][(Point[0].y)/2]==0)
      return;
  Blood=Blood+Window ->AddBlood * 2;
  int inX=(Point[0].x)/2;
  int inY=(Point[0].y)/2;
  Window ->Grass[inX][inY]=0;
  Window ->GrassNumber--;
  return;
}

TWormList :: TWormList(TScrsvWindow * Window)
{
  Window = Window;
  HeadWorm = new TWorm(Window, TWormGene(), 800, TPoint());
  ThisWorm = HeadWorm;
  for (int i=1;i<=Window ->WormNumber;i++)
  {
      TPoint ThisPoint;
      TWormGene ThisGene;
      ThisGene.LeftProb=1+random(50);
      ThisGene.RightProb=1+random(50);
      ThisPoint.x=random(MAXX);
      ThisPoint.y=random(MAXY);

      ThisWorm ->NextWorm = new TWorm(Window,ThisGene,800,ThisPoint);
      ThisWorm = ThisWorm ->NextWorm;
  }
  ThisWorm ->NextWorm=0;
  PreWorm = HeadWorm;
  ThisWorm= PreWorm ->NextWorm;
}

TWormList :: ~TWormList()
{
  ThisWorm = HeadWorm ->NextWorm;
  while(ThisWorm)
  {
```

```
        HeadWorm ->NextWorm = ThisWorm ->NextWorm;
        delete ThisWorm;
        ThisWorm = HeadWorm ->NextWorm;
    }
    delete HeadWorm;
    Window ->WormNumber = 0;
}

TWorm * TWormList :: GetAWorm()
{
    if(ThisWorm==0)
    {
        PreWorm = HeadWorm;
        ThisWorm= PreWorm ->NextWorm;
    }
    else
    {
        PreWorm = ThisWorm;
        ThisWorm= ThisWorm ->NextWorm;
    }
    return ThisWorm;
}

BOOL TWormList :: AddWorm(TWorm * AWorm)
{
    if(AWorm==0) return false;
    AWorm ->NextWorm = HeadWorm ->NextWorm;
    HeadWorm ->NextWorm = AWorm;
    Window ->WormNumber++;
    if(PreWorm ->NextWorm! =ThisWorm)
    {
        PreWorm = PreWorm ->NextWorm;
    }
    return true;
}

BOOL TWormList :: DelWorm()
{
    if(ThisWorm==0) return false;
    PreWorm ->NextWorm = ThisWorm ->NextWorm;
    delete ThisWorm;
    ThisWorm = PreWorm ->NextWorm;
    Window ->WormNumber--;
```

```
    return true;
  }

/*    Project scrsv
        FILE:            scrsvabd.cpp
        Source file for implementation of scrsvAboutDlg (TDialog).
**************************************
*/
#include <owl\pch.h>
#if ! defined(__FLAT__)
#endif
#include "tscrsvap.h"
#include "scrsvabd.h"

scrsvAboutDlg::scrsvAboutDlg (TWindow *parent, TResId resId, TModule *module)
       : TDialog(parent, resId, module)
{}
scrsvAboutDlg::~scrsvAboutDlg ()
{       Destroy();   }

/*    Project scrsv
        FILE:            tscrsvap.cpp
        Source file for implementation of TScrsvApp (TApplication).
************************************
*/#include <owl\pch.h>
#include "tscrsvap.h"
#include "tscrsvwn.h"                       // Definition of client class.
#include "tsetpdlg.h"

//{{TScrsvApp Implementation}}
//////////////////////////////////////
// TScrsvApp
// =====
//
TScrsvApp::TScrsvApp () : TApplication("屏幕保护")
{
}

TScrsvApp::~TScrsvApp ()
{
}

//////////////////////////////////////
```

```
// TScrsvApp
// =====
// Application intialization.
//
void TScrsvApp::InitMainWindow ()
{
        TScrsvWindow *frame = new TScrsvWindow(0, 0, 0);
        SetMainWindow(frame);
}

int TScrsvApp::Run ()
{
  if (strcmp(lpCmdLine, "/c") == 0 || strcmp(lpCmdLine, "-c") == 0)
  {
        ConfigureDialog = new TSetupDlg(0, ID_SETUP, this);
        HWND deskAppletHWnd = ::FindWindow(WC_DIALOG, "Desktop");
        TWindow * deskAppletWnd = 0;
        if (deskAppletHWnd)
            deskAppletWnd = new TWindow(deskAppletHWnd);
        if (deskAppletWnd)
            ConfigureDialog->SetParent(deskAppletWnd);
        ConfigureDialog->Execute();
        delete ConfigureDialog;
        delete deskAppletWnd;
        return Status;
  }
  else return TApplication::Run();
}

int OwlMain (int , char * [])
{
  TScrsvApp App;
  App.EnableBWCC();
  return App.Run();
}

/*   Project scrsv
    FILE:           tsetpdlg.cpp
    Source file for implementation of TSetupDlg (TDialog).
*************************************
*/
#include <owl\pch.h>
#include <stdio.h>
```

```
#include "tsetpdlg.h"
#include "scrsvabd.h"
#include "define.h"

DEFINE_RESPONSE_TABLE1(TSetupDlg, TDialog)
  EV_COMMAND(IDHELP, CmHelp),
END_RESPONSE_TABLE;

//{{TSetupDlg Implementation}}
TSetupDlgXfer::TSetupDlgXfer()
{
  char settings[4];
  AddGrass.LowValue = 0;
  AddGrass.HighValue = 100;
  ::GetPrivateProfileString(ScrnSectionStr, SetGrassStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &(AddGrass.Position));

  AddBlood.LowValue = 0;
  AddBlood.HighValue = 100;
  ::GetPrivateProfileString(ScrnSectionStr, SetBloodStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &(AddBlood.Position));
}

TSetupDlgXfer TSetupDlg::SetupData;

TSetupDlg::TSetupDlg (TWindow * parent, TResId resId, TModule * module):
      TDialog(parent, resId, module)
{
//{{TSetupDlgXFER_USE}}
      AddBlood = new TScrollBar(this, IDC_ADDBLOOD);
      AddGrass = new TScrollBar(this, IDC_ADDGRASS);
      SetTransferBuffer(&SetupData);
//{{TSetupDlgXFER_USE_END}}
}

TSetupDlg::~TSetupDlg ()
{
      Destroy();
}

void TSetupDlg::CmHelp()
{
  scrsvAboutDlg aDlg(this) ;
```

```
      aDlg. Execute();
  }

  void TSetupDlg::CloseWindow (int retVal){
    char settings[4];
    wsprintf(settings, "%d", AddGrass->GetPosition());
    ::WritePrivateProfileString(ScrnSectionStr, SetGrassStr, settings, "control.ini");
    wsprintf(settings, "%d", AddBlood->GetPosition());
    ::WritePrivateProfileString(ScrnSectionStr, SetBloodStr, settings, "control.ini");
    TDialog::CloseWindow(retVal);
  }

  /*   Project scrsv
        FILE:            tscrsvwn.cpp
        Source file for implementation of TScrsvWindow (TWindow).
  *****************************************************************
  */
  #include <owl\pch.h>
  #include "tscrsvap.h"
  #include "define.h"
  #include "tscrsvwn.h"
  #include "tworm.h"
  #include <stdio.h>

  int MAXX;
  int MAXY;

  DEFINE_RESPONSE_TABLE1(TScrsvWindow, TFrameWindow)
  //{{TScrsvWindowRSP_TBL_BEGIN}}
        EV_WM_ACTIVATE,
        EV_WM_LBUTTONDOWN,
        EV_WM_RBUTTONDOWN,
        EV_WM_ACTIVATEAPP,
        EV_WM_MBUTTONDOWN,
        EV_WM_SYSCOMMAND,
        EV_WM_SYSKEYDOWN,
        EV_WM_MOUSEMOVE,
        EV_WM_KEYDOWN,
        EV_WM_TIMER,
  //{{TScrsvWindowRSP_TBL_END}}
  END_RESPONSE_TABLE;

  //{{TScrsvWindow Implementation}}
```

```
///////////////////////////////////////////
// TScrsvWindow
// ========
// Construction/Destruction handling.
TScrsvWindow::TScrsvWindow (TWindow * parent, const char far * title, TModule * module)
      : TFrameWindow(parent, title, 0, false, module)
{
  ShowCursor(FALSE);
  Attr.Style = WS_POPUP;
  Attr.ExStyle = WS_EX_TOPMOST;
  SetBkgndColor(BackColor);
}

void TScrsvWindow::EvActivate (uint active, bool, HWND)
{
  if(! active)
      PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvLButtonDown (uint, TPoint&)
{
  PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvRButtonDown (uint, TPoint&)
{
  PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvActivateApp (bool active, HTASK )
{
  if(! active)
      PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvMButtonDown (uint, TPoint& )
{
  PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvSysCommand (uint cmdType, TPoint& )
{
  if ((cmdType & 0xFFF0) ! = SC_SCREENSAVE)
```

```
        DefaultProcessing();
}

void TScrsvWindow::EvSysKeyDown (uint, uint , uint )
{
  PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvMouseMove (uint, TPoint& point)
{
  if (point != PrevPt)
        PostMessage(WM_CLOSE);
}

void TScrsvWindow::EvKeyDown (uint , uint , uint )
{
  PostMessage(WM_CLOSE);
}

void TScrsvWindow::GetWindowClass (WNDCLASS& wndClass)
{
  TWindow::GetWindowClass(wndClass);
  wndClass.hIcon = 0;
  wndClass.style |= CS_SAVEBITS;
  wndClass.hbrBackground = (HBRUSH)GetStockObject(NULL_BRUSH);
}

void TScrsvWindow::SetupWindow ()
{
        TWindow::SetupWindow();

  GetCursorPos(PrevPt);
  TRect r;
  ::GetWindowRect(GetDesktopWindow(), &r);
  MoveWindow(r, TRUE);

  MAXX = r.Width();
  MAXY = r.Height();

  char settings[4];

  ::GetPrivateProfileString(ScrnSectionStr, SetGrassStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &AddGrass);
```

```
  ::GetPrivateProfileString(ScrnSectionStr, SetBloodStr, "50", settings, sizeof(settings), "control.ini");
  sscanf(settings, "%d", &AddBlood);

  WormNumber=20+random(20);
  WormList = new TWormList(this);

  Grass = new uint8 * [MAXX/2+1];            // STEP 1: SET UP THE ROWS.
  for (int i = 0; i <= MAXX/2; i++)
      Grass[i] = new uint8[MAXY/2+1];        // STEP 2: SET UP THE COLUMNS
  for(i=0; i<=MAXX/2; i++)
  for(int j=0; j<=MAXY/2; j++)
      Grass[i][j] = 0;

  GrassNumber=0;
  int GN = 300+random(250);
  for (i=1;i<=GN;i++)
  {
          TPoint NewPoint;
          NewPoint.x=random(MAXX/2);
          NewPoint.y=random(MAXY/2);
          if (Grass[NewPoint.x][NewPoint.y]==0)
          {
            GrassNumber++;
            Grass[NewPoint.x][NewPoint.y]=1;
          }
  }

  SetTimer(TIME_WORM,100);
  DC = new TClientDC( * this) ;
}

TScrsvWindow::~TScrsvWindow ()
{
  Destroy();
  ShowCursor(TRUE);
  for (int i = 0; i <= MAXX/2; i++)
          delete[] Grass[i];                  // STEP 1: DELETE THE COLUMNS
  delete[] Grass;               // STEP 2: DELETE THE ROWS
  KillTimer(TIME_WORM);
  delete DC;
  delete WormList;
}
```

```
void TScrsvWindow::EvTimer (uint timerId)
{
  TWindow::EvTimer(timerId);
  DrawGrass();
  DrawWorm();
  return;
}

void TScrsvWindow::DrawGrass()
{
  int dg;
  dg=(int)(AddGrass/10);
  for (int i=1;i<=dg;i++)
  {
      int ThisX,ThisY;
      ThisX=1+random(MAXX/2);
      ThisY=1+random(MAXY/2);
      if(Grass[ThisX][ThisY]! =1)
      {
          Grass[ThisX][ThisY]=1;
          GrassNumber++;
          DC->SetPixel(ThisX * 2,ThisY * 2,TColor::LtGreen);
      }
  }
}

void TScrsvWindow::DrawWorm()
{
  TWorm * ThisWorm = WormList->GetAWorm();
  TWorm::TResult AResult;
  while(ThisWorm)
  {
      DC->SetPixel(ThisWorm->Point[4],TColor(BackColor));
      AResult=ThisWorm->Run();
      DC->SetPixel(ThisWorm->Point[0],TColor(RGB(255,255 - ThisWorm->Blood/4,255 - This-
Worm->Blood/4)));
      if(AResult= =TWorm::Dead)
      {
          for(int i=0;i<=4;i++)
            DC->SetPixel(ThisWorm->Point[i],TColor(BackColor));
          WormList->DelWorm();
          if (WormNumber<1)
```

```
                KillTimer(TIME_WORM);
            }
            if(AResult==TWorm::Birth)
            {
                TWorm *ChildWorm1, *ChildWorm2;
                ChildWorm1=new TWorm(this, ThisWorm->Gene,ThisWorm->Blood,ThisWorm->Point
[0]);
                ChildWorm2=new TWorm(this, ThisWorm->Gene,ThisWorm->Blood,ThisWorm->Point
[0]);
                WormList->AddWorm(ChildWorm1);
                WormList->AddWorm(ChildWorm2);
                for(int i=0;i<=4;i++)
                  DC->SetPixel(ThisWorm->Point[i],TColor(BackColor));
                WormList->DelWorm();
            }
            ThisWorm = WormList->GetAWorm();
        }
    }

    void TScrsvWindow::Paint (TDC& dc, bool erase, TRect& rect)
    {
      TWindow::Paint(dc, erase, rect);
      for (int X=1;X<=MAXX/2;X++)
      for (int Y=1;Y<=MAXY/2;Y++)
          if (Grass[X][Y]! =0)
              dc.SetPixel(X*2,Y*2,TColor::LtGreen);
    }

;--------------------------------------------
;  SUBSYSTEM:    scrsv.exe Module Defintion File
;  FILE:         tscrsvap.def
;--------------------------------------------
NAME scrsv
DESCRIPTION 'SCRNSAVE :Worm Screen Saver'
EXETYPE     WINDOWS
CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE
HEAPSIZE    4096
STACKSIZE   8192
```

第 6 章

单文档界面和多文档界面

单文档界面(SDI)是相对多文档界面(MDI)而言的。如果在一个主窗口中可以同时打开若干个文档窗口，我们就称它为多文档界面(MDI)，否则就是单文档界面。SDI 与 MDI 结构的应用程序框架都可以用 AppExpert 工具来构造，在构造时可以选择是否采用 Doc\View(文档与视图)结构。采用 Doc\View 结构可以方便地创建具有复杂的文档结构、多种文档类型以及视图类型的应用程序。本章着重讨论 SDI 与 MDI 中各种窗口管理的方法，所给实例也只限于对位图文件(bmp)的显示并不对文档进行加工，所以没有采用 Doc\View 结构，依然采用 Frame\Client 结构。

6.1 单文档位图观察器

单文档界面通常采用 TFrameWindow 类和 TDecoratedFrame 类的派生类作为应用程序的主窗口的框架窗口类，它们包括了一个具有特定功能的客户窗口类，由框架窗口完成大部分窗口操作的任务(比如，改变大小、显示或改变标题、设置窗口风格等)，客户窗口则完成应用程序的实际任务(比如，绘图等)。关于 TFrameWindow 类作为主窗口的框架窗口类，我们在前面已有所讨论，而且也比较简单;TDecoratedFrame 类作为经过修饰的框架窗口类我们也接触过，所以在本节中我们结合一个“位图观察器”的例子来学习单文档界面的窗口组织结构。

使用 AppExpert 生成一个普通的单文档应用程序框架，主窗口采用 TDecoratedFrame 的派生类 TSDIDecFrame:

```
TSDIDecFrame * frame = new TSDIDecFrame(0, GetName(), 0, true);
```

通过调用 SetupSpeedBar 函数在主窗口中插入包含一系列小按钮的控制条:

```
void TSdiviewApp::SetupSpeedBar(TDecoratedFrame * frame)
{
  ApxHarbor = new THarbor( * frame);

  TDockableControlBar * cb = new TDockableControlBar(frame);
  cb->Insert( * new TButtonGadget(CM_FILENEW, CM_FILENEW));
  cb->Insert( * new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN));
  cb->Insert( * new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
```

```
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
    cb->Insert( * new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
    cb->Insert( * new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
    cb->Insert( * new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));

    cb->SetCaption("Toolbar");
    cb->SetHintMode(TGadgetWindow::EnterHints);
    ApxHarbor->Insert( * cb, alTop);
}
```

小按钮和小描述窗口在 Windows 应用程序中是很常见的，比如在 Borland C++的集成环境中我们可以使用一系列方便的小按钮用来代替寻找和点击菜单的麻烦；小按钮还带有一个小描述窗口，它是黄底黑字，鼠标所到之处，光标尾部都弹出简洁的解释。我们把那些通常放置在菜单条下面的小按钮、位图和其他小的控制统称为小器件。TGadget 类是各种小器件类的基类(其中包括小按钮类)，它的各种派生类见表 6-1。

表 6-1

派生类	描　　述
TBitmapGadget	显示一个位图
TButtonGadget	位图小按钮
TControlGadget	封装了控制，比如编辑框或组合列表框，使其可以插入小器件窗口
TTextGadget	小文本框
TSeparatorGadget	划分小器件的逻辑组

TGadget 类的构造函数和析构函数是：

```
TGadget(int id = 0,          TBorderStyle = None);
virtual ~TGadget();
```

TGadget 类的对象属于一个小器件窗口，它具有边框、边界和自己的坐标系统。你可以选择边框风格或者设置边框并重载成员函数 PaintBorder 来创建你自己的小器件边框。如果你改变了边框、边界或边框风格，小器件窗口的 GadgetChangedSize 数据成员就被调用。

小器件窗口作为活动窗口可以接收鼠标事件。当小器件窗口接收到了消息后，它判断哪个小器件应该接收这个消息，如何调用成员函数来处理消息。

小按钮类 TButtonGadget 类作为 TGadget 类的派生类，提供用户可以开关的小按钮。你也可以设置小按钮的属性参数，比如，颜色、风格和形状等。

TButtonGadget 类的构造函数：

```
TButtonGadget(TResId bmpResId,
              int id,
              TType type = Command,
              bool enabled = false,
              TState state = Up,
              bool repeat = false);
```

通常，小按钮分为命令按钮和属性按钮两类。属性按钮包括了 radio 按钮(排他的)和检查按钮(不排他的)，数据成员 TType 枚举了按钮的种类。

TButtonGadget 类包括了一系列函数来改变按钮的风格：SetAntialiasEdges 函数用来设置开关按钮；SetNotchCorners 函数用来控制角切割；SetShadowStyle 函数用来改变按钮的阴影风格。

TButtonGadget 对象以下面的方式响应鼠标事件：当鼠标按键按下时，小按钮按下；当鼠标按键释放时，小按钮释放。当鼠标按钮的状态为“up”时命令才传入。当鼠标按钮被按下时，TButtonGadget 对象获得了鼠标的控制权，接收鼠标在当前窗口的全部消息。当鼠标按钮抬起时，小按钮释放鼠标控制权给当前窗口。

小按钮的 bmpResId 代表小按钮的显示位图，我们通常见到的是打开文件、打印文件等位图，位图的定义在资源文件中，它们的含义一看就知道了，这也是小按钮与菜单命令相比所具有的优势之一。ID 代表小按钮标识值，它也是小按钮发出的命令值。为了方便起见，我们经常将标识小按钮位图的 ID 值和标识小按钮命令的 ID 值设置为相同的值。在程序中响应小按钮的命令与响应菜单项的命令非常相似，而且通常我们的程序中都具有菜单项和小按钮两种选择方式，因此程序中相关的小按钮和菜单项命令通常是一致的。

定义应用程序可以接受的文档类型为 BMP，在程序中说明为：

```
FileData.Flags=OFN_FILEMUSTEXIST|OFN_HIDEREADONLY| OFN_OVERWRITEPROMPT;
  FileData.SetFilter("Bmp Files (*.bmp)|*.bmp|");
  FileData.DefExt = "bmp";
#if defined(BI_PLAT_WIN32)
  if (TSystem::IsNT() || TSystem::IsWin95())
    FileData.Flags |= OFN_LONGNAMES;
  if (TSystem::IsWin95())
    FileData.Flags |= OFN_EXPLORER;
#endif
```

在单文档位图观察器程序中设计了一个可以观察位图的窗口类 TSdiviewWindow 类，它是 TWindow 类的派生类；设计了一个 TDecoratedFrame 类的派生类 TTSDIDecFrame；设计了一个应用程序类 TSdiviewApp，它是 TApplication 类的派生类。在 TSdiviewApp 类的成员函数 InitMainWindow 中，构造了一个 TTSDIDecFrame 类的对象，将它作为应用程序主窗口(的框架窗口)：

```
void TSdiviewApp::InitMainWindow()
{
```

```
    if (nCmdShow != SW_HIDE)
      nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;
    TTSDIDecFrame * frame = new TTSDIDecFrame(0, GetName(), 0, true);
    frame->SetIcon(this, IDI_SDIAPPLICATION);
    frame->SetIconSm(this, IDI_SDIAPPLICATION);
    frame->AssignMenu(IDM_SDI);
    frame->Attr.AccelTable = IDM_SDI;
    SetupSpeedBar(frame);
    TStatusBar * sb = new TStatusBar(frame, TGadget::Recessed,
                                     TStatusBar::CapsLock       |
                                     TStatusBar::NumLock         |
                                     TStatusBar::ScrollLock);
    frame->Insert(*sb, TDecoratedFrame::Bottom);
    SetMainWindow(frame);
    frame->SetMenuDescr(TMenuDescr(IDM_SDI));
}
```

TTSDIDecFrame 类的构造函数将 TSdiviewWindow 类作为窗口的缺省客户窗口，这样，TSdiviewWindow 类窗口就成为应用程序主窗口的客户窗口：

```
TSDIDecFrame::TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool
trackMenuSelection, TModule * module)
  :   TDecoratedFrame(parent, title, !clientWnd ? new TSdiviewWindow(0, "") : clientWnd, trackMenuS-
election, module)
  { }
```

TSdiviewWindow 类是一个可以显示 BMP 位图的窗口类，它派生于 TWindow 类，它的类型说明如下：

```
class TSdiviewWindow : public TWindow {
  public:
    TSdiviewWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TSdiviewWindow();
      BOOL     ReplaceWith(const char * );

  public:      TDib *      Dib;
      int      PixelWidth;
      int      PixelHeight;

//{{TSdiviewWindowVIRTUAL_BEGIN}}
  public:
    virtual void Paint(TDC& dc, bool erase, TRect& rect);
//{{TSdiviewWindowVIRTUAL_END}}
};    //{{TSdiviewWindow}}
```

其中，Dib 是一个“与设备无关”的位图 TDib 的指针;PixelWidth 和 PixelHeight 表示像素点的

宽度和高度；函数 LoadBitmapFile 为窗口装载一个位图文件。TSdiviewWindow 类的构造函数如下：

```
TSdiviewWindow::TSdiviewWindow(TWindow * parent, const char far * title, TModule * module)
:    TWindow(parent, title, module)
{
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;
  Dib = 0;
  Scroller = new TScroller(this, 1, 1, 200, 200);
}
```

为了显示较大的位图，窗口的风格中增加了水平和垂直的滚动条。ReplaceWith 函数装载了指定文件名的位图文件：

```
BOOL TSdiviewWindow::ReplaceWith(const char * name)
{
  Invalidate();
  TDib * newDib;
  try {
    newDib = new TDib(name);
  }
  catch (TXGdi) {
    MessageBox("Cannot open bitmap file", GetApplication()->GetName(), MB_OK);
    return false;
  }
  delete Dib;
  Dib = newDib;
  PixelWidth = Dib->Width();
  PixelHeight = Dib->Height();
  return true;
}
```

其中，try {…} catch (…) {…}结构是当程序在 try 段出现异常时调用 catch 的处理方法；通过构造一个“与设备无关”的位图对象，对象中包含了指定位图文件中的位图；如果位图中没有调色板，就使用系统缺省的调色板，并通知用户。

Paint 函数实现了在窗口中显示位图的功能：

```
void TSdiviewWindow::Paint(TDC& dc, bool erase, TRect& rect)
{
  TWindow::Paint(dc, erase, rect);

  TRect clientRect = GetClientRect();
  if (Dib)
  {
    TRect imageRect(0,0, PixelWidth, PixelHeight);
```

```
        clientRect += TPoint((int)Scroller->XPos, (int)Scroller->YPos);
        dc.SetDIBitsToDevice(imageRect, TPoint(0,0), *Dib);
        dc.PatBlt(TRect(TPoint(PixelWidth,0), clientRect.BottomRight()));
        dc.PatBlt(TRect(TPoint(0,PixelHeight), clientRect.BottomRight()));
    } else
        dc.PatBlt(clientRect, PATCOPY);
}
```

其中，SetDIBitsToDevice 函数和 PatBlt 函数将"与设备无关"的位图显示在窗口中。

图 6-1 是单文档的位图观察器，图 6-2 给出的则是该程序的项目文件。

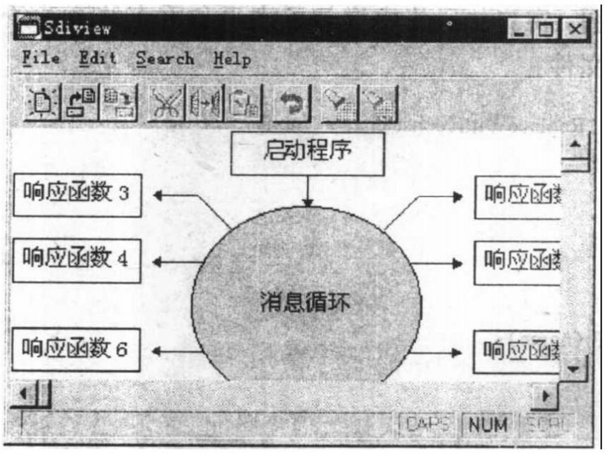


图 6-1　单文档的位图观察器

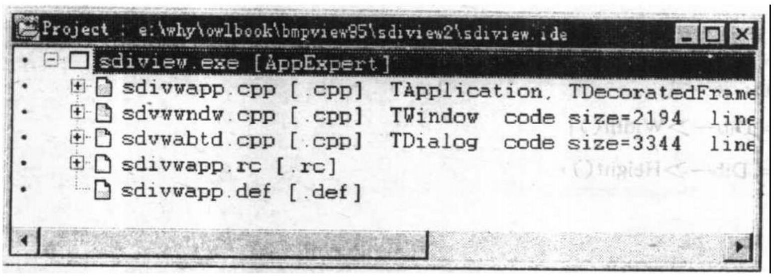


图 6-2　单文档位图观察器应用程序项目文件

下面是单文档应用程序代码：

```
//----------------------------------------------------------------------
//  FILE:            SdivwApp.h
//  Class definition for TSdiviewApp (TApplication).
//----------------------------------------------------------------------
#if !defined(sdivwapp_h)            // Sentry, use file only if it's not already included.
#define sdivwapp_h
#include <owl/controlb.h>
#include <owl/docking.h>
#include <owl/opensave.h>
#include <owl/rcntfile.h>
```

```
#include "SdivwApp.rh"              // Definition of all resources.
//
// FrameWindow must be derived to override Paint for Preview and Print.
//
//{{TDecoratedFrame = TSDIDecFrame}}
class TSDIDecFrame : public TDecoratedFrame {
  public:
    TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool trackMenuSelection = false, TModule * module = 0);
      ~TSDIDecFrame();
};    //{{TSDIDecFrame}}

//{{TApplication = TSdiviewApp}}
class TSdiviewApp : public TApplication, public TRecentFiles {
  private:
    void SetupSpeedBar(TDecoratedFrame * frame);
  public:
    TSdiviewApp();
    virtual ~TSdiviewApp();

    TOpenSaveDialog::TData FileData;          // Data to control open/saveas standard dialog.
    void OpenFile(const char * fileName = 0);
    THarbor *         ApxHarbor;
//{{TSdiviewAppVIRTUAL_BEGIN}}
  public:
    virtual void InitMainWindow();
//{{TSdiviewAppVIRTUAL_END}}
//{{TSdiviewAppRSP_TBL_BEGIN}}
  protected:
    void CmFileNew();
    void CmFileOpen();
    void CmHelpAbout();
    int32 CmFileSelected(uint wp, int32 lp);
//{{TSdiviewAppRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TSdiviewApp);
};    //{{TSdiviewApp}}
#endif     // sdivwapp_h sentry.

//---------------------------------------------------------------------------
//  FILE:          SdvwWndw.h
//  Class definition for TSdiviewWindow (TWindow).
//---------------------------------------------------------------------------
#if !defined(sdvwwndw_h)              // Sentry, use file only if it's not already included.
```

```
#define sdvwwndw_h
#include "SdivwApp.rh"              // Definition of all resources.

//{{TWindow = TSdiviewWindow}}
class TSdiviewWindow : public TWindow {
  public:
    TSdiviewWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TSdiviewWindow();
      BOOL     ReplaceWith(const char * );
  public:
      TDib *      Dib;
      int       PixelWidth;
      int       PixelHeight;
//{{TSdiviewWindowVIRTUAL_BEGIN}}
  public:
    virtual void Paint(TDC& dc, bool erase, TRect& rect);
//{{TSdiviewWindowVIRTUAL_END}}
};    //{{TSdiviewWindow}}
#endif        // sdvwwndw_h sentry.

//-----------------------------------------------------------------
//  FILE:           SdivwApp.cpp
//  Source file for implementation of TSdiviewApp (TApplication).
//-----------------------------------------------------------------
#include <owl/pch.h>
#include <owl/buttonga.h>
#include <owl/statusba.h>
#include <stdio.h>
#include "SdivwApp.h"
#include "SdvwWndw.h"                          // Definition of client class.
#include "SdvwAbtD.h"                         // Definition of about dialog.

//{{TSdiviewApp Implementation}}
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE2(TSdiviewApp, TRecentFiles, TApplication)
//{{TSdiviewAppRSP_TBL_BEGIN}}
  EV_COMMAND(CM_FILENEW, CmFileNew),
  EV_COMMAND(CM_FILEOPEN, CmFileOpen),
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
  EV_REGISTERED(MruFileMessage, CmFileSelected),
```

```
//{{TSdiviewAppRSP_TBL_END}}
END_RESPONSE_TABLE;

//-------------------------------------
// TSdiviewApp
//
TSdiviewApp::TSdiviewApp() : TApplication("Sdiview"), TRecentFiles(".\\Sdiview.ini", 4)
{
  // Common file file flags and filters for Open/Save As dialogs. Filename and directory are
  // computed in the member functions CmFileOpen, and CmFileSaveAs.
  //
  FileData.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT;
  FileData.SetFilter("Bmp Files (*.bmp)|*.bmp|");
  FileData.DefExt = "bmp";
#if defined(BI_PLAT_WIN32)
  if (TSystem::IsNT() || TSystem::IsWin95())
    FileData.Flags |= OFN_LONGNAMES;

  if (TSystem::IsWin95())
    FileData.Flags |= OFN_EXPLORER;
#endif
}

TSdiviewApp::~TSdiviewApp()
{
}

void TSdiviewApp::SetupSpeedBar(TDecoratedFrame* frame)
{
  ApxHarbor = new THarbor(*frame);
  // Create default toolbar New and associate toolbar buttons with commands.
  //
  TDockableControlBar* cb = new TDockableControlBar(frame);
  cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW));
  cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN));
  cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
  cb->Insert(*new TSeparatorGadget(6));
  cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
  cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
  cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
  cb->Insert(*new TSeparatorGadget(6));
  cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
  cb->Insert(*new TSeparatorGadget(6));
```

```
    b->Insert( * new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
    cb->Insert( * new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));
    // Add caption and fly-over help hints.
    //
    cb->SetCaption("Toolbar");
    cb->SetHintMode(TGadgetWindow::EnterHints);
    ApxHarbor->Insert( * cb, alTop);
}
//-------------------------------------------
// TSdiviewApp
// ~~~~
// Application intialization.
//
void TSdiviewApp::InitMainWindow()
{
    if (nCmdShow != SW_HIDE)
      nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

    TSDIDecFrame * frame = new TSDIDecFrame(0, GetName(), 0, true);
    // Assign ICON w/ this application.
    //
    frame->SetIcon(this, IDI_SDIAPPLICATION);
    frame->SetIconSm(this, IDI_SDIAPPLICATION);
    // Menu associated with window and accelerator table associated with table.
    //
    frame->AssignMenu(IDM_SDI);
    // Associate with the accelerator table.
    //
    frame->Attr.AccelTable = IDM_SDI;
    SetupSpeedBar(frame);
    TStatusBar * sb = new TStatusBar(frame, TGadget::Recessed,
                                     TStatusBar::CapsLock        |
                                     TStatusBar::NumLock          |
                                     TStatusBar::ScrollLock);
    frame->Insert( * sb, TDecoratedFrame::Bottom);
    SetMainWindow(frame);
    frame->SetMenuDescr(TMenuDescr(IDM_SDI));
}

//-------------------------------------------
// TSdiviewApp
// ~~~~~~~~~~~
// Menu File New command
```

```
//
void TSdiviewApp::CmFileNew()
{}

//------------------------------------------------------------------
// TSdiviewApp
// ~~~~~~~~~
// Menu File Open command
//
void TSdiviewApp::CmFileOpen()
{
*FileData.FileName = 0;
  TSdiviewWindow * client = TYPESAFE_DOWNCAST(GetMainWindow()->GetClientWindow(), TS-
diviewWindow);        // Client window for the frame.
  if (client->CanClose())
    if (TFileOpenDialog(GetMainWindow(), FileData).Execute() == IDOK)
      OpenFile();
}

void TSdiviewApp::OpenFile(const char * fileName)
{
  if (fileName)
    strcpy(FileData.FileName, fileName);

  TSdiviewWindow * ClientWindow = TYPESAFE_DOWNCAST(MainWindow->GetClientWindow(),
TSdiviewWindow);
  ClientWindow->ReplaceWith(FileData.FileName);
  // Add the file to the MRU list.
  //
  SaveMenuChoice(FileData.FileName);
}

//{{TSDIDecFrame Implementation}}
TSDIDecFrame::TSDIDecFrame(TWindow * parent, const char far * title, TWindow * clientWnd, bool
trackMenuSelection, TModule * module)
:    TDecoratedFrame(parent, title, !clientWnd ? new TSdiviewWindow(0, "") : clientWnd, trackMenuS-
election, module)
{ }

TSDIDecFrame::~TSDIDecFrame()
{}

//------------------------------------------------------------------
```

```
// TSdiviewApp
// ~~~~~~~~~
// Menu Help About Sdiview command
//
void TSdiviewApp::CmHelpAbout()
{
  // Show the modal dialog.
  //
  TSdiviewAboutDlg(MainWindow).Execute();
}

int32 TSdiviewApp::CmFileSelected(uint wp, int32)
{
  TAPointer<char> text = new char[_MAX_PATH];
  GetMenuText(wp, text, MAX_PATH);
  OpenFile(text);
  return 0;
}

int OwlMain(int , char * [])
{
  TSdiviewApp app;
  return app.Run();
}

//-----------------------------------------------------------
//  FILE:         SdvwWndw.cpp
//  Source file for implementation of TSdiviewWindow (TWindow).
//-----------------------------------------------------------
#include <owl/pch.h>
#include <owl/scroller.h>
#include "SdivwApp.h"
#include "SdvwWndw.h"
#include <stdio.h>

//{{TSdiviewWindow Implementation}}
//------------------------------------------
// TSdiviewWindow
// ~~~~~~~~
// Construction/Destruction handling.
//
TSdiviewWindow::TSdiviewWindow(TWindow * parent, const char far * title, TModule * module)
:    TWindow(parent, title, module)
```

```
{
    Attr.Style |= WS_VSCROLL | WS_HSCROLL;
    Dib = 0;
    Scroller = new TScroller(this, 1, 1, 200, 200);
}

TSdiviewWindow::~TSdiviewWindow()
{
    Destroy();
    delete Dib;
}

BOOL TSdiviewWindow::ReplaceWith(const char * name)
{
    Invalidate();
    TDib * newDib;
    try {   newDib = new TDib(name);
    }
    catch (TXGdi) {
        MessageBox("Cannot open bitmap file", GetApplication()->GetName(), MB_OK);
        return false;
    }
    delete Dib;
    Dib = newDib;
    PixelWidth = Dib->Width();
    PixelHeight = Dib->Height();
    return true;
}

void TSdiviewWindow::Paint(TDC& dc, bool erase, TRect& rect)
{
    TWindow::Paint(dc, erase, rect);

    TRect clientRect = GetClientRect();
    if (Dib)
    {   TRect imageRect(0,0, PixelWidth, PixelHeight);
        clientRect += TPoint((int)Scroller->XPos, (int)Scroller->YPos);
        dc.SetDIBitsToDevice(imageRect, TPoint(0,0), *Dib);
        dc.PatBlt(TRect(TPoint(PixelWidth,0), clientRect.BottomRight()));
        dc.PatBlt(TRect(TPoint(0,PixelHeight), clientRect.BottomRight()));
    } else
        dc.PatBlt(clientRect, PATCOPY);
}
```

6.2　多文档位图观察器

使用 OWL 构造一个多文档界面需要如下几个类或它们的派生类：

TApplication 类	应用程序类
TMDIFrame 类	MDI 主窗口框架窗口类
TMDIClient 类	MDI 主窗口客户窗口类
TMDIChild 类	MDI 子窗口框架窗口类
TWindow 类	子窗口的客户窗口

前面已讨论过，通常应用程序中的一个窗口是由分别完成不同功能的两个窗口（框架窗口和客户窗口）组成的。表面看起来，MDI 界面只需要两种窗口，一种是 MDI 主窗口，一种是 MDI 子窗口，实际上程序中需要四个不同的窗口类。

TMDIFrame 是多文档界面的主窗口框架窗口，它自动处理创建和初始化 MDI 客户窗口（TMDIClient 对象）。缺省时，TMDIFrame 设置窗口风格为 WS_CLIPCHILDREN，这样在刷新 MDI 框架窗口和主窗口背景时就不会出现明显的闪烁。

多文档界面（MDI）的客户窗口类管理着由 TMDIFrame 建立的框架窗口的 MDI 子窗口。

TMDIChild 类作为 MDI 子窗口的框架窗口定义了 MDI 子窗口的基本行为。子窗口可以在父窗口的客户区域中创建。因为子窗口存在于父窗口并被限制于父窗口的边界中，所以在定义子窗口之前先要定义父窗口。要使用 MDI 子窗口，子窗口类一定要是 TMDIChild 类的派生类。

MDI 子窗口的客户窗口通常是能够完成一些设计任务（比如，绘图）的窗口。它可以是 TWindow 类的派生类。

在应用程序类的 InitMainWindow 函数中构造并设置应用程序的 MDI 主窗口，包括框架窗口 TDecoratedMDIFrame 类的对象和客户窗口 TMdiviewMDIClient 类的对象，还在主窗口中增加了状态条和控件条：

```
void TMdiviewApp::InitMainWindow()
{
  if (nCmdShow != SW_HIDE)
    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;
  MdiClient = new TMdiviewMDIClient;
  TDecoratedMDIFrame * frame = new TDecoratedMDIFrame(Name, IDM_MDI, *MdiClient, true);
  frame->SetIcon(this, IDI_MDIAPPLICATION);
  frame->SetIconSm(this, IDI_MDIAPPLICATION);

  frame->Attr.AccelTable = IDM_MDI;
  SetupSpeedBar(frame);
  TStatusBar * sb = new TStatusBar(frame, TGadget::Recessed,
                                   TStatusBar::CapsLock       |
```

```
                    TStatusBar::NumLock        |
                    TStatusBar::ScrollLock);
  frame->Insert(*sb, TDecoratedFrame::Bottom);
  SetMainWindow(frame);
  frame->SetMenuDescr(TMenuDescr(IDM_MDI));
}
```

其中，TDecoratedMDIFrame 类是 TMDIFrame 类的派生类，它可以安排一些窗口的配件(比如，控件条和状态条)；TMdiviewMDIClient 类是 TMDIClient 类的派生类。

主窗口的客户窗口类 TMdiviewMDIClient 类负责新建、打开和关闭一个文件：

```
DEFINE_RESPONSE_TABLE1(TMdiviewMDIClient, TMDIClient)
//{{TMdiviewMDIClientRSP_TBL_BEGIN}}
  EV_COMMAND(CM_MDIFILENEW, CmFileNew),
  EV_COMMAND(CM_MDIFILEOPEN, CmFileOpen),
  EV_COMMAND(CM_FILECLOSE, CmFileClose),
//{{TMdiviewMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;
```

其中，打开文件的方法与 SDI 中的方法有些不同，见下：

```
void TMdiviewMDIClient::CmFileOpen()
{
  TMdiviewApp * theApp = TYPESAFE_DOWNCAST(GetApplication(), TMdiviewApp);
  *(theApp->FileData.FileName) = 0;
  if (TFileOpenDialog(this, theApp->FileData).Execute() == IDOK)
    OpenFile();
}

void TMdiviewMDIClient::OpenFile(const char * fileName)
{
  TMdiviewApp * theApp = TYPESAFE_DOWNCAST(GetApplication(), TMdiviewApp);
  if (fileName)
    strcpy(theApp->FileData.FileName, fileName);
  TMdiviewMDIChild * child = new TMdiviewMDIChild(*this, theApp->FileData.FileName, new TM-
diviewWindow(0));

  child->SetIcon(GetApplication(), IDI_DOC);
  child->SetIconSm(GetApplication(), IDI_DOC);
  TMdiviewMDIChild * curChild = (TMdiviewMDIChild *)GetActiveMDIChild();
  if (curChild && (curChild->GetWindowLong(GWL_STYLE) & WS_MAXIMIZE))
    child->Attr.Style |= WS_MAXIMIZE;
  child->Create();
  TMdiviewWindow * ClientWindow = TYPESAFE_DOWNCAST(child->GetClientWindow(), TMdi-
viewWindow);
```

```
    ClientWindow->ReplaceWith(theApp->FileData.FileName);
    theApp->SaveMenuChoice(theApp->FileData.FileName);
}
```

上面的程序中创建了一个 TMdiviewMDIChild 类的对象作为 MDI 子窗口框架窗口，而使 TMdiviewWindow 类的对象作为这个子窗口的客户窗口。

```
TMdiviewMDIChild::TMdiviewMDIChild(TMDIClient& parent, const char far * title, TWindow * clientWnd, bool shrinkToClient, TModule * module)
  :    TMDIChild(parent, title, ! clientWnd ? new TMdiviewWindow(0, "") : clientWnd, shrinkToClient, module)
{ }
```

当子窗口的框架窗口关闭后，子窗口的客户窗口也就随之关闭。

作为多文档界面框架窗口的 TDecoratedMDIFrame 类具有窗口修饰功能，可以在其上增加状态条和控件条等。成员函数 SetupSpeedBar 在控件条上增加了小按钮：

```
void TMdiviewApp::SetupSpeedBar(TDecoratedMDIFrame * frame)
{
    ApxHarbor = new THarbor( * frame);

    // Create default toolbar New and associate toolbar buttons with commands.
    //
    TDockableControlBar * cb = new TDockableControlBar(frame);
    cb->Insert( * new TButtonGadget(CM_MDIFILENEW, CM_MDIFILENEW));
    cb->Insert( * new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
    cb->Insert( * new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
    cb->Insert( * new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
    cb->Insert( * new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
    cb->Insert( * new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));

    // Add caption and fly-over help hints.
    //
    cb->SetCaption("Toolbar");
    cb->SetHintMode(TGadgetWindow::EnterHints);

    ApxHarbor->Insert( * cb, alTop);
}
```

图 6-3 是我们设计的多文档位图观察器，图 6-4 则是该程序的项目文件。

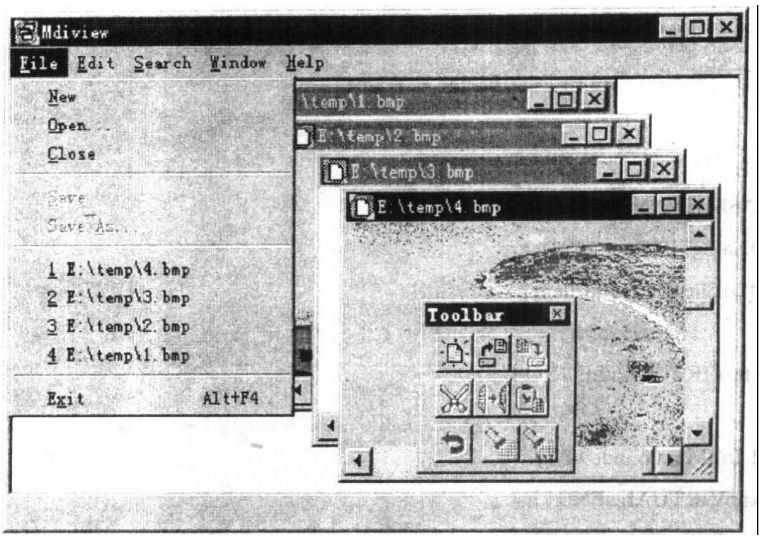


图 6-3 多文档的位图观察器

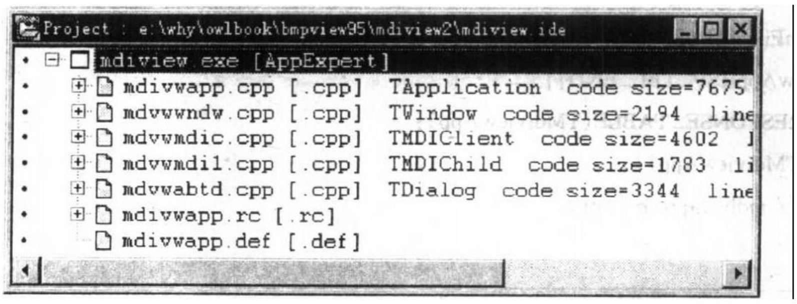


图 6-4 多文档位图观察器应用程序项目文件

程序中使用了一系列小按钮，它们的位图通常都比较简单，在编程序的中间搞一搞“艺术创作”也是很有趣的事情。

下面是多文档应用程序的部分代码：

```
//----------------------------------------------------------------------
// FILE:            MdivwApp.h
// Class definition for TMdiviewApp (TApplication).
//----------------------------------------------------------------------
#if !defined(mdivwapp_h)              // Sentry, use file only if it's not already included.
#define mdivwapp_h


#include <owl/controlb.h>
#include <owl/docking.h>
#include <owl/opensave.h>
#include <owl/rcntfile.h>
#include "MdvwMDIC.h"
#include "MdivwApp.rh"            // Definition of all resources.


//{{TApplication = TMdiviewApp}}
```

```
class TMdiviewApp : public TApplication, public TRecentFiles {
  private:
    void SetupSpeedBar(TDecoratedMDIFrame * frame);
  public:
    TMdiviewApp();
    virtual ~TMdiviewApp();
    TOpenSaveDialog::TData FileData;
    TMdiviewMDIClient * MdiClient;
    THarbor *       ApxHarbor;
//{{TMdiviewAppVIRTUAL_BEGIN}}
  public:
    virtual void InitMainWindow();
//{{TMdiviewAppVIRTUAL_END}}
//{{TMdiviewAppRSP_TBL_BEGIN}}
  protected:
    void CmHelpAbout();
    int32 CmFileSelected(uint wp, int32 lp);
//{{TMdiviewAppRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TMdiviewApp);
};    //{{TMdiviewApp}}
#endif    // mdivwapp_h sentry.


//-------------------------------------------------------------
//  FILE:           MdvwWndw.h
//  Class definition for TMdiviewWindow (TWindow).
//-------------------------------------------------------------
#if ! defined(mdvwwndw_h)            // Sentry, use file only if it's not already included.
#define mdvwwndw_h


#include "MdivwApp.rh"             // Definition of all resources.

//{{TWindow = TMdiviewWindow}}
class TMdiviewWindow : public TWindow {
  public:
    TMdiviewWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);
    virtual ~TMdiviewWindow();
      BOOL     ReplaceWith(const char * );
  public:
      TDib *      Dib;
      int        PixelWidth;
      int        PixelHeight;
//{{TMdiviewWindowVIRTUAL_BEGIN}}
  public:
```

```
    virtual void Paint(TDC& dc, bool erase, TRect& rect);
//{{TMdiviewWindowVIRTUAL_END}}
};    //{{TMdiviewWindow}}
#endif    // mdvwwndw_h sentry.

//----------------------------------------------------------------------
//  FILE:          MdvwMDIC.h
//  Class definition for TMdiviewMDIClient (TMDIClient).
//----------------------------------------------------------------------
#if ! defined(mdvwmdic_h)          // Sentry, use file only if it's not already included.
#define mdvwmdic_h
#include "MdivwApp.rh"            // Definition of all resources.

//{{TMDIClient = TMdiviewMDIClient}}
class TMdiviewMDIClient : public TMDIClient {
  public:
    int    ChildCount;                    // Number of child window created.
    TMdiviewMDIClient(TModule * module = 0);
    virtual ~TMdiviewMDIClient();
    void OpenFile(const char * fileName = 0);
//{{TMdiviewMDIClientVIRTUAL_BEGIN}}
  protected:
    virtual void SetupWindow();
//{{TMdiviewMDIClientVIRTUAL_END}}
//{{TMdiviewMDIClientRSP_TBL_BEGIN}}
  protected:
    void CmFileNew();
    void CmFileOpen();
    void CmFileClose();
//{{TMdiviewMDIClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TMdiviewMDIClient);
};    //{{TMdiviewMDIClient}}
#endif    // mdvwmdic_h sentry.

//----------------------------------------------------------------------
//  FILE:          MdvwMDI1.h
//  Class definition for TMdiviewMDIChild (TMDIChild).
//----------------------------------------------------------------------
#if ! defined(mdvwmdi1_h)          // Sentry, use file only if it's not already included.
#define mdvwmdi1_h
#include "MdivwApp.rh"            // Definition of all resources.

//{{TMDIChild = TMdiviewMDIChild}}
```

```
class TMdiviewMDIChild : public TMDIChild {
  public:
    TMdiviewMDIChild(TMDIClient& parent, const char far * title, TWindow * clientWnd, bool shrinkTo-
Client = false, TModule * module = 0);
    virtual ~TMdiviewMDIChild();
};      //{{TMdiviewMDIChild}}
#endif      // mdvwmdi1_h sentry.

//------------------------------------------------------------------------
//  FILE:           MdivwApp.cpp
//  Source file for implementation of TMdiviewApp (TApplication).
//------------------------------------------------------------------------
#include <owl/pch.h>
#include <owl/buttonga.h>
#include <owl/statusba.h>

#include "MdivwApp.h"
#include "MdvwMDIC.h"
#include "MdvwAbtD.h"                  // Definition of about dialog.

//{{TMdiviewApp Implementation}}
//
// Build a response table for all messages/commands handled by the application.
//
DEFINE_RESPONSE_TABLE2(TMdiviewApp, TRecentFiles, TApplication)
//{{TMdiviewAppRSP_TBL_BEGIN}}
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
  EV_REGISTERED(MruFileMessage, CmFileSelected),
//{{TMdiviewAppRSP_TBL_END}}
END_RESPONSE_TABLE;

//----------------------------------------------------------
// TMdiviewApp
// ~~~~
//
TMdiviewApp::TMdiviewApp() : TApplication("Mdiview"), TRecentFiles(".\\Mdiview.ini", 4)
{}

TMdiviewApp::~TMdiviewApp()
{}

void TMdiviewApp::SetupSpeedBar(TDecoratedMDIFrame * frame)
{
```

```
    ApxHarbor = new THarbor( * frame);
    // Create default toolbar New and associate toolbar buttons with commands.
    //
    TDockableControlBar * cb = new TDockableControlBar(frame);
    cb->Insert( * new TButtonGadget(CM_MDIFILENEW, CM_MDIFILENEW));
    cb->Insert( * new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
    cb->Insert( * new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
    cb->Insert( * new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
    cb->Insert( * new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
    cb->Insert( * new TSeparatorGadget(6));
    cb->Insert( * new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
    cb->Insert( * new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));
    // Add caption and fly-over help hints.
    //
    cb->SetCaption("Toolbar");
    cb->SetHintMode(TGadgetWindow::EnterHints);
    ApxHarbor->Insert( * cb, alTop);
}

//----------------------------------
// TMdiviewApp
// Application intialization.
//
void TMdiviewApp::InitMainWindow()
{
   if (nCmdShow != SW_HIDE)
     nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;
   MdiClient = new TMdiviewMDIClient;
   TDecoratedMDIFrame * frame = new TDecoratedMDIFrame(Name, IDM_MDI, * MdiClient, true);
   // Assign ICON w/ this application.
   //
   frame->SetIcon(this, IDI_MDIAPPLICATION);
   frame->SetIconSm(this, IDI_MDIAPPLICATION);
   // Associate with the accelerator table.
   //
   rame->Attr.AccelTable = IDM_MDI;
   SetupSpeedBar(frame);
   TStatusBar * sb = new TStatusBar(frame, TGadget::Recessed,
                                    TStatusBar::CapsLock        |
```

```
                                TStatusBar::NumLock          |
                                TStatusBar::ScrollLock);
  frame->Insert( * sb, TDecoratedFrame::Bottom);
  SetMainWindow(frame);
  frame->SetMenuDescr(TMenuDescr(IDM_MDI));
}

//-------------------------------------------------
// TMdiviewApp
// ~~~~~~~
// Menu Help About Mdiview command
//
void TMdiviewApp::CmHelpAbout()
{
TMdiviewAboutDlg(MainWindow).Execute();
}

int32 TMdiviewApp::CmFileSelected(uint wp, int32)
{
  TAPointer<char> text = new char[_MAX_PATH];

  GetMenuText(wp, text, MAX_PATH);

  TMdiviewMDIClient * theClient
    = TYPESAFE_DOWNCAST(GetMainWindow()->GetClientWindow(), TMdiviewMDIClient);
  if (theClient)
    theClient->OpenFile(text);
  return 0;
}

int OwlMain(int , char * [])
{
  TMdiviewApp app;
  return app.Run();
}

//-------------------------------------------------------
// FILE:          MdvwWndw.cpp
// Source file for implementation of TMdiviewWindow (TWindow).
//-------------------------------------------------------
#include <owl/pch.h>
#include <owl/scroller.h>
#include "MdivwApp.h"
```

```
#include "MdvwWndw.h"

#include <stdio.h>

//{{TMdiviewWindow Implementation}}
//--------------------------------
// TMdiviewWindow
// ~~~~~~~~
// Construction/Destruction handling.
//
TMdiviewWindow::TMdiviewWindow(TWindow* parent, const char far* title, TModule* module)
: TWindow(parent, title, module)
{
  Attr.Style |= WS_VSCROLL | WS_HSCROLL;
  Dib = 0;
  Scroller = new TScroller(this, 1, 1, 200, 200);
}

TMdiviewWindow::~TMdiviewWindow()
{
  Destroy();
  delete Dib;
}

BOOL TMdiviewWindow::ReplaceWith(const char* name)
{
  Invalidate();
  TDib* newDib;
  try {
    newDib = new TDib(name);
  }
  catch (TXGdi) {
    MessageBox("Cannot open bitmap file", GetApplication()->GetName(), MB_OK);
    return false;
  }
  delete Dib;
  Dib = newDib;
  PixelWidth = Dib->Width();
  PixelHeight = Dib->Height();
  return true;
}

void TMdiviewWindow::Paint(TDC& dc, bool erase, TRect& rect)
```

```
{
  TWindow::Paint(dc, erase, rect);
  TRect clientRect = GetClientRect();
  if (Dib)
  {
      TRect imageRect(0,0, PixelWidth, PixelHeight);
      clientRect += TPoint((int)Scroller->XPos, (int)Scroller->YPos);
      dc.SetDIBitsToDevice(imageRect, TPoint(0,0), *Dib);
      dc.PatBlt(TRect(TPoint(PixelWidth,0), clientRect.BottomRight()));
      dc.PatBlt(TRect(TPoint(0,PixelHeight), clientRect.BottomRight()));
  } else
      dc.PatBlt(clientRect, PATCOPY);
}

//-------------------------------------------------------------------
// FILE:          MdvwMDIC.cpp
// Source file for implementation of TMdiviewMDIClient (TMDIClient).
//-------------------------------------------------------------------
#include <owl/pch.h>
#include <owl/listbox.h>
#include <stdio.h>
#include "MdivwApp.h"
#include "MdvwWndw.h"
#include "MdvwMDI1.h"
#include "MdvwMDIC.h"

//{{TMdiviewMDIClient Implementation}}
//
// Build a response table for all messages/commands handled
// by TMdiviewMDIClient derived from TMDIClient.
//
DEFINE_RESPONSE_TABLE1(TMdiviewMDIClient, TMDIClient)
//{{TMdiviewMDIClientRSP_TBL_BEGIN}}
  EV_COMMAND(CM_MDIFILENEW, CmFileNew),
  EV_COMMAND(CM_MDIFILEOPEN, CmFileOpen),
  EV_COMMAND(CM_FILECLOSE, CmFileClose),
//{{TMdiviewMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;

//-------------------------------------------------------------
// TMdiviewMDIClient
// Construction/Destruction handling.
//
```

```
TMdiviewMDIClient::TMdiviewMDIClient(TModule * module)
 : TMDIClient(module)
{
  ChildCount = 0;
}

TMdiviewMDIClient::~TMdiviewMDIClient()
{
  Destroy();
}

//------------------------------------------------
// TMdiviewMDIClient
// MDIClient site initialization.
//
void TMdiviewMDIClient::SetupWindow()
{
  // Default SetUpWindow processing.
  //
  TMDIClient::SetupWindow();
  TMdiviewApp * theApp = TYPESAFE_DOWNCAST(GetApplication(), TMdiviewApp);
  // Common file file flags and filters for Open/Save As dialogs. Filename and directory are
  // computed in the member functions CmFileOpen, and CmFileSaveAs.
  //
  theApp->FileData.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY | OFN_OVER-
WRITEPROMPT;
  theApp->FileData.SetFilter("bmp Files (*.bmp)|*.bmp|All Files (*.*)|*.*|");
  theApp->FileData.DefExt = "bmp";
 #if defined(BI_PLAT_WIN32)
  if (TSystem::IsNT() || TSystem::IsWin95())
    theApp->FileData.Flags |= OFN_LONGNAMES;
  if (TSystem::IsWin95())
    theApp->FileData.Flags |= OFN_EXPLORER;
 #endif
}

//------------------------------------------------
// TMdiviewMDIClient
// Menu File New command
//
void TMdiviewMDIClient::CmFileNew()
{
  TAPointer<char> title = new char[255];
```

```
    TAPointer<char> numberStr = new char[16];
    // Generate a title for the MDI child window.
    //
    if (GetModule()->LoadString(IDS_UNTITLED, title, sizeof title)) {
      sprintf(numberStr, " - %d", ++ChildCount);
      strcat(title, numberStr);
    }
    else
      title[0] = '\0';
    TMdiviewMDIChild * child = new TMdiviewMDIChild( * this, title, 0);
    // Associate ICON w/ this child window.
    //
    child->SetIcon(GetApplication(), IDI_DOC);
    child->SetIconSm(GetApplication(), IDI_DOC);
    // If the current active MDI child is maximize then this one should be also.
    //
    TMdiviewMDIChild * curChild = (TMdiviewMDIChild * )GetActiveMDIChild();
    if (curChild && (curChild->GetWindowLong(GWL_STYLE) & WS_MAXIMIZE))
      child->Attr.Style |= WS_MAXIMIZE;
    child->Create();
}

void TMdiviewMDIClient::OpenFile(const char * fileName)
{
    TMdiviewApp * theApp = TYPESAFE_DOWNCAST(GetApplication(), TMdiviewApp);
    if (fileName)
      strcpy(theApp->FileData.FileName, fileName);
    // Create a MDIChild window whose client is TMdiviewWindow.
    //
    TMdiviewMDIChild * child = new TMdiviewMDIChild( * this, theApp->FileData.FileName, new TM-
diviewWindow(0));
    // Associate ICON with this child window.
    //
    child->SetIcon(GetApplication(), IDI_DOC);
    child->SetIconSm(GetApplication(), IDI_DOC);
    // If the current active MDI child is maximize then this one should be also.
    //
    TMdiviewMDIChild * curChild = (TMdiviewMDIChild * )GetActiveMDIChild();
    if (curChild && (curChild->GetWindowLong(GWL_STYLE) & WS_MAXIMIZE))
      child->Attr.Style |= WS_MAXIMIZE;
    child->Create();
    TMdiviewWindow * ClientWindow = TYPESAFE_DOWNCAST(child->GetClientWindow(), TMdi-
viewWindow);
```

```
    ClientWindow->ReplaceWith(theApp->FileData.FileName);
    // Add the file to the MRU list.
    //
    theApp->SaveMenuChoice(theApp->FileData.FileName);
  }

//--------------------------------
// TMdiviewMDIClient
// Menu File Open command
//
void TMdiviewMDIClient::CmFileOpen()
{
   TMdiviewApp * theApp = TYPESAFE_DOWNCAST(GetApplication(), TMdiviewApp);
   // Display standard Open dialog box to select a file name.
   //
   *(theApp->FileData.FileName) = 0;
   if (TFileOpenDialog(this, theApp->FileData).Execute() == IDOK)
     OpenFile();
}

//--------------------------------
// TMdiviewMDIClient
// Menu File Close command
//
void TMdiviewMDIClient::CmFileClose()
{
  TMdiviewMDIChild * curChild = (TMdiviewMDIChild *)GetActiveMDIChild();

  if (curChild) {
    curChild->CloseWindow();
  }
}

//--------------------------------------
//  FILE:          MdvwMDI1.cpp
//  Source file for implementation of TMdiviewMDIChild (TMDIChild).
//--------------------------------------
#include <owl/pch.h>
#include "MdivwApp.h"
#include "MdvwWndw.h"
#include "MdvwMDI1.h"

//{{TMdiviewMDIChild Implementation}}
```

```
//------------------------------------------------------------
// TMdiviewMDIChild
// ~~~~~~~~
// Construction/Destruction handling.
//
TMdiviewMDIChild::TMdiviewMDIChild (TMDIClient& parent, const char far * title, TWindow * clientWnd, bool shrinkToClient, TModule * module)
:    TMDIChild(parent, title, ! clientWnd ? new TMdiviewWindow(0, "") : clientWnd, shrinkToClient, module)
{ }
TMdiviewMDIChild::~TMdiviewMDIChild()
{
  Destroy();
}
```

附录 A

ObjectWindows 5.0 类库

A.1　TWindow 类

头文件：window.h

描述：TWindow 是 TEventHandler 和 TStreamableBase 的子类，它封装了窗口类的特性和与窗口类有关的一系列成员函数，提供窗口的初始化、创建、操作及关闭窗口功能。

TWindow 是最基本的窗口类，它可以变化大小或移动。你可以直接用 TWindow 来定义一个窗口对象，但是通常不这样做，我们往往只把它作为一个更有个性的窗口类的基类。TWindow 是所有的窗口类的基类，包括 TFrameWindow，TControl，TDialog 和 TMDIChild 类，它在 DWL 5.0 中的位置如图 A-1 所示。

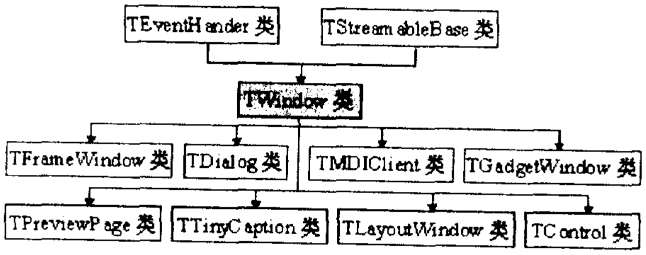


图 A-1　TWindow 类在 OWL 5.0 中的位置

Public 数据成员：

TWindowAttr Attr;

定义窗口属性。

WNDPROC DefaultProc;

定义缺省的窗口过程的地址。

HWND HWindow;

定义了和窗口对象相关的界面元素的句柄。

TWindow * Parent;

指向窗口对象的父窗口对象。

TScroller * Scroller;

指向窗口滚动条对象。

TStatus Status;

标志窗口对象初始化时的错误。

char far * Title;

指向窗口对象的标题。

Public 构造函数和析构函数:

TWindow(HWND hWnd, TModule * module = 0);

给一个非 OWL 窗口构造一个 TWindow 类作为别名。

TWindow(TWindow * parent, const char far * title = 0, TModule * module = 0);

若有父窗口 parent 为 this;若无父窗口 parent 为 0。

virtual ~TWindow();

窗口类的析构函数。

Public 成员函数:

static void AdjustWindowRect(TRect& rect, uint32 style, bool menu);

根据客户区域调节窗口大小。

static void AdjustWindowRectEx(TRect&, rect, uint32 style, bool menu, uint32 exStyle);

计算包含扩展属性的窗口大小。

void BringWindowToTop();

激活一个弹出窗口或子窗口，并将它显示在一系列窗口的最前面。

virtual bool CanClose();

决定是否关闭窗口，返回 ture 则关闭，否则不关闭。

void CheckDlgButton(int buttonId, uint check);

在按钮上增加(或去掉)检验标记: 如果 check 不是 0, 检验标记就移到下一个按钮上;如果 check 是 0, 检验标记就清除;对于三态按钮，check 的值可以是 0(清除)、1(标记)、2(灰色)。

void CheckRadioButton(int firstButtonId, int lastButtonId, int checkButtonId);

在一个组里，给 checkButtonId 代表的 Radio 按钮增加检验标记，同时去掉组里其它按钮的检验标记。firstButtonId 和 lastButtonId 代表组里的首尾两个按钮。

void ChildBroadcastMessage(uint msg, WPARAM wParam, LPARAM lParam);

在所有的立即子窗口中发出消息。

HWND ChildWindowFromPoint(const TPoint&) const;

返回窗口中 TPoint& 所指一点所属的子窗口句柄。如果返回 0, 说明这一点不在窗口区域里。

TWindow * ChildWithId(int id) const;

返回具有特定 id 值的子窗口指针。

void ClearFlag(TWindowFlag mask);

在数据成员 Flags 中清除特定的标志，比如 wfAlias 和 wfTransfer 等。

void ClientToScreen(TPoint& point) const;

将窗口客户坐标系上的点坐标转化为屏幕坐标系上的点坐标。

virtual void CloseWindow(int retval = 0);

关闭窗口。在真正关闭以前，确认是否可以关闭: 如果窗口是主窗口就调用 GetApplication->CanClose 函数; 否则调用 this->CanClose 函数。

void CmExit();

响应具有 CM_EXIT 这样 ID 值的菜单项。如果是主窗口，调用 CloseWindow。

virtual bool Create();

创建一个与窗口对象相关的界面元素，成功则返回 true;失败返回 false。

void CreateCaret(HBITMAP hBitmap);

HBITMAP 提供插字符位图的形状。

void CreateCaret(bool isGray, int width, int height);

为系统创建新的特殊形状的插字符。

bool CreateChildren();

在主窗口表中创建子窗口，成功返回 true。

LRESULT DefaultProcessing();

提供缺省的窗口过程。

virtual LRESULT DefWindowProc(uint msg, WPARAM wParam, LPARAM lParam);

提供缺省的窗口消息处理函数。

virtual void Destroy(int retVal = 0);

删除一个相关的窗口界面元素。

static void DestroyCaret();

检查当前窗口所拥有的插字符，并把它从屏幕上删除掉。

void DisableAutoCreate();

关闭子窗口的界面元素随父窗口建立和显示的功能。

void DisableTransfer();

关闭窗口元素与传送缓冲区的数据交换功能。

virtual LRESULT Dispatch(TEventInfo& info, WPARAM wp, LPARAM lp = 0);

将消息分发给相应的响应成员函数。

void DragAcceptFiles(bool accept);

如果窗口可以拖动一个文件，就把 accept 设置为 true。

void DrawMenuBar();

重新绘制菜单条。

void EnableAutoCreate();

打开子窗口的界面元素随父窗口建立和显示的功能。

void EnableScrollBar(uint sbFlags = SB_BOTH, uint arrowFlags = ESB_ENABLE_BOTH);

打开或关闭窗口的滚动条功能。sbFlags 设置滚动条的内容如 SB_CTL、SB_HORZ、SB_VERT、或 SB_BOTH 等。arrowFlags 设置打开或关闭特定的滚动条如 ESB_ENABLE_BOTH、ESB_DISABLE_LTUP、ESB_DISABLE_RTDN 或 ESB_DISABLE_BOTH 等。

void EnableTransfer();

打开窗口元素与传送缓冲区的数据交换功能。

virtual bool EnableWindow(bool enable);

打开窗口，使它可以接收来自鼠标器和键盘的事件。

int EnumProps(PROPENUMPROC proc);

将事件队列里的相一个一个的放到回调函数 proc 中。

void EvChildInvalid(HWND hWnd);

响应子窗口编辑控件发出的消息 WM_CHILDINVALID。

virtual LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);

窗口回调函数响应 WM_COMMAND 消息的响应函数。id 值是菜单项或控件对象的值；hWndCtl 保存了发出消息的控件对象的句柄，如果消息不是从控件对象发出的，值为 0；notifyCode 保存了控件的通报消息值，如果消息是从一个加速键表发出的，值为 1，如果从菜单发出的，值为 0。

void EvCommandEnable(TCommandEnabler& ce);

窗口回调函数响应 WM_COMMAND_ENABLE 消息的响应函数。调用 CmXxxx 函数来响应输入的消息。

void EvSysCommand (uint cmdType, TPoint& point);

响应用户使用系统菜单和系统按钮时发出的消息，表 A-1 列出了这些消息的意义。

表 A-1

系统消息	意义解释
SC_CLOSE	关闭窗口
SC_HOTKEY	激活特定的窗口
SC_HSCROLL	滚动条横向滚动
SC_KEYMENU	通过击键得到菜单
SC_MAXIMIZE（或 SC_ZOOM）	窗口最大化
SC_MINIMIZE（或 SC_ICON）	窗口最小化
SC_MOUSEMENU	通过鼠标器得到菜单
SC_NEXTWINDOW	移动到下一个窗口
SC_PREVWINDOW	移动到上一个窗口
SC_SCREENSAVE	执行特定的屏幕保护程序
SC_SIZE	窗口大小设置
SC_TASKLIST	激活任务管理器
SC_VSCROLL	滚动条纵向滚动

TWindow * FirstThat(TCondFunc test, void * paramList = 0);

为子窗口表中的每一个子窗口调用非窗口类成员测试函数。若测试函数返回 true 则停止测试并返回测试对象的指针，否则返回 0。

TWindow * FirstThat(TCondMemFunc test, void * paramList = 0);

为子窗口表中的每一个子窗口调用窗口类成员测试函数。若测试函数返回 true 则停止测试并返回测试对象的指针，否则返回 0。

bool FlashWindow(bool invert);

将窗口在激活和非激活状态之间转化。如果 invert 不是 0，窗口就闪烁；如果是 0，窗口就按照初始的激活状态设置。

void ForEach(TActionFunc action, void * paramList = 0);

调用指定的非窗口类成员函数，并返回子窗口中的所有成员。

void ForEach(TActionMemFunc action, void * paramList = 0);

调用指定的窗口类成员函数，并返回子窗口中的所有成员。

LRESULT ForwardMessage(bool send = true);

前推窗口当前的消息。如果 send 的值是 true，调用 SendMessage 函数；否则调用 PostMessage 函数。

LRESULT ForwardMessage(HWND hWnd, bool send = true);

前推其它窗口当前的消息。如果 send 的值是 true，调用 SendMessage 函数；否则调用 PostMessage 函数。

static HWND GetActiveWindow();

返回当前活动窗口的句柄。

TApplication * GetApplication()const;

返回指向和 this 相关的 TApplication 对象的指针。

static HWND GetCapture();

返回当前获得鼠标器资源的窗口句柄。

static uint GetCaretBlinkTime();

返回插字符闪烁的时间。

static void GetCaretPos(TPoint& point);

返回插字符在当前窗口客户区的坐标值。

long GetClassLong(int index) const;

返回关于窗口类的 32 位信息。

uint16 GetClassWord(int index) const;

返回关于窗口风格的 16 位信息。

TRect GetClientRect() const;

返回窗口客户区的坐标值。

void GetClientRect(TRect& rect) const;

获得窗口客户区的坐标值并将其复制到 Trect 定义的变量中。

static void GetCursorPos(TPoint& pos);

获得光标在当前窗口的位置坐标。

static HWND GetDesktopWindow();

返回当前桌面窗口的句柄。

int GetDlgCtrlID() const;

返回控件对象的 ID 值。

HWND GetDlgItem(int childId) const;

返回 ID 值指定的控件对象的句柄。

uint GetDlgItemInt(int childId, bool * translated = 0,bool isSigned = true) const;

返回 ID 值指定的控件对象。

uint GetDlgItemText(int childId, char far * text, int max) const;

获得 ID 值指定的控件对象的文本。

TWindow * GetFirstChild()

返回界面对象子窗口表中的第一个子窗口的指针。

static HWND GetFocus()

获得当前焦点窗口的句柄。

void GetHWndState();

把已存在的 HWnd 句柄中的风格、坐标系以及 ID 值(不包括标题)拷贝到一个窗口对象中。

int GetId();

返回窗口的 ID 值。

TModule * GetModule() const;

返回窗口的模块对象指针。

HWND GetLastActivePopup() const;

返回子窗口表中最后一个激活的弹出窗口的句柄。

TWindow * GetLastChild();

返回界面对象子窗口表中的最后一个子窗口的指针。

HMENU GetMenu() const;

返回窗口对象的菜单对象。如果没有菜单则返回 0。

HWND GetNextDlgGroupItem(HWND hWndCtrl, bool previous = false) const;

返回对话框中的下一个或上一个控件对象的句柄。previous 是 0 则返回下一个控件；若是 1 则返回上一个控件。

HWND GetNextDlgTabItem(HWND HWndCtrl, bool previous = false) const;

返回对话框中的用 TAB 键切换的下一个或上一个控件对象的句柄。previous 是 0 则返回下一个控件；若是 1 则返回上一个控件。

HWND GetNextWindow(uint dirFlag) const;

返回窗口管理器中下一个窗口的句柄。

TWindow * GetWindowPtr(HWND hWnd);

在应用程序中使用这个函数，返回指定的窗口对象的指针。

HWND GetParent() const;

返回窗口的父窗口的句柄，如果没有父窗口则返回 0。

HANDLE GetProp(uint16 atom) const;

返回窗口性能表的句柄。

HANDLE GetProp(const char far * string) const;

返回窗口性能表的句柄。

int GetScrollPos(int bar) const;

返回滚动条目前的值。

void GetScrollRange(int bar, int& minPos, int& maxPos) const;

返回滚动条的取值范围。

static HWND GetSysModalWindow();

返回系统模块窗口的句柄。

HMENU GetSystemMenu(bool revert = false) const;

返回窗口的系统菜单。

WNDPROC GetThunk()const;

返回实例转化的回调函数。

HWND GetTopWindow()const;

返回窗口中最上面的子窗口的句柄。

bool GetUpdateRect(TRect& rect, bool erase = true) const;

获得窗口更新区域在屏幕上的坐标。

bool GetUpdateRgn(TRegion& rgn, bool erase = true) const;

将窗口在屏幕上的更新区域拷贝到 rgn 中。

HWND GetWindow(uint cmd) const;

返回窗口的句柄。

HFONT GetWindowFont();

返回控件对象所有的字体。如果使用系统缺省的字体，返回 NULL。

long GetWindowLong(int index) const;

根据 index 值返回关于窗口的信息，表 A-2 列出了 index 的取值和意义。

表 A－2

取　　值	意　　义
GWL_EXSTYLE	扩展窗口风格
GWL_STYLE	窗口风格
GWL_WNDPROC	窗口回调函数地址
DWL_DLGPROC	对话框回调函数地址　　（对话框特有）
DWL_MSGRESULT	对话框信息的取值　　（对话框特有）
DWL_USER	附加信息　　（对话框特有）

bool GetWindowPlacement(WINDOWPLACEMENT * place) const;

获得显示和放置的信息。

void GetWindowRect(TRect& rect) const;

获得窗口所占屏幕的区域。

TRect GetWindowRect() const;

返回窗口所占屏幕的区域。

HTASK GetWindowTask() const;

返回建立特定窗口的任务指针。

int GetWindowText(char far * string, int maxCount) const;

获得窗口标题字符串。

int GetWindowTextLength() const;

获得窗口标题字符串的长度。

void GetWindowTextTitle();

用窗口当前的标题更新窗口对象中的数据成员。

uint16 GetWindowWord(int index) const;

根据 index 值返回关于窗口的信息，表 A－3 列出了 index 的取值和意义。

表 A－3

取　　值	意　　义
GWW_HINSTANCE	拥有窗口的模块实例指针
GWW_HWNDPARENT	父窗口句柄
GWW_ID	子窗口的 ID 数

LRESULT HandleMessage(uint msg, WPARAM wParam = 0, LPARAM lParam = 0);

给窗口传送消息。

void HideCaret();

在屏幕上隐藏插字符。

bool HiliteMenuItem(HMENU hMenu, uint idItem, uint hilite);

设置菜单项加亮或不加亮。idItem 指出要加亮的菜单项，hilite 值的意义列于表 A－4 中。

表 A-4

取　　值	意　　义
MF_BYCOMMAND	idItem 参数包含菜单项的 ID 值
MF_BYPOSITION	idItem 参数包含菜单项的相对位置
MF_HILITE	加亮菜单项
MF_UNHILITE	不加亮菜单项

virtual bool HoldFocusHWnd(HWND hWndLose, HWND hWndGain);

响应子窗口的一个请求，当它失去焦点时保存它的句柄。

operator HWND()const;

将窗口类指针转化成窗口句柄，使它可以在标准 Windows API 中使用。

virtual bool IdleAction(long idleCount);

当没有消息是窗口执行此函数。idleCount 指出两次消息中加入空闲消息的次数。

void Invalidate(bool erase = true);

使窗口客户区域失效，令其重画。

void InvalidateRect(const TRect&, bool erase = true);

使窗口客户区域的由 TRect& 定义一个特定区域失效，令其重画。

void InvalidateRgn(HRGN hRgn, bool erase = true);

使窗口客户区域的由 HRGN hRgn 定义一个特定区域失效，使其重画。

bool IsChild(HWND) const;

返回指定窗口是否是一个子窗口。

uint IsDlgButtonChecked(int buttonId) const;

返回指定的按钮的检验状态。如果返回 0，则按钮是非检验状态；如果返回 1，按钮是检验状态；如果按钮返回 3，按钮是不确定状态。

bool IsFlagSet(TWindowFlag mask);

返回窗口对象的位标志是否设置。

bool IsIconic() const;

返回窗口是否是图标型或是最小化状态。

bool IsWindow() const;

当使用了窗口句柄是返回 1。

bool IsWindowEnabled() const;

返回窗口是否活动。

bool IsWindowVisible() const;

返回窗口是否可见。

bool IsZoomed() const;

返回窗口是否在最大化状态。

bool KillTimer(uint timerId);

确定窗口中指定的计时器。

bool LockWindowUpdate();

锁定一个窗口。

void MapWindowPoints(HWND hWndTo, TPoint * points, int count) const;

从一个窗口到另外一个窗口的对应转化。

int MessageBox(const char far * text,const char far * caption = 0, uint type = MB_OK);

创建并显示一个对话框，其中包括一条信息、一个标题、图标和按钮。如果标题的值为 0，就显示缺省的标题，按钮中可以包括 MB_Xxxx 这样的组合按钮。函数返回值和意义列于表 A-5 中。如果使用 BWCC，消息框就是 BWCC 风格的；如果使用 CTRL 3D，消息框就是 CTRL 3D 风格的；如果既使用 BWCC 又使用 CTRL 3D，消息框就是标准风格的。

表 A-5

取　　值	意　　义
IDABORT	用户选择 ABORT 按钮
IDCANCEL	用户选择 CANCEL 按钮
IDIGNORE	用户选择 IGNORE 按钮
IDNO	用户选择 NO 按钮
IDOK	用户选择 OK 按钮
IDRETRY	用户选择 RETRY 按钮
IDYES	用户选择 YES 按钮

void MoveWindow(int x, int y, int w, int h, bool repaint = false);

按照所给坐标和大小移动窗口。

void MoveWindow(const TRect& rect, bool repaint = false);

按照所给区域移动窗口。

TWindow * Next();

返回子窗口表中下一个兄弟窗口的指针。

unsigned NumChildren();

返回窗口中子窗口的个数。

TClipboard& OpenClipboard();

打开剪贴板，这时其它窗口中的内容不能再改变。如果调用失败，说明剪贴板正由其它窗口使用。

virtual void Paint(TDC& dc, bool erase, TRect& rect);

响应消息 WM_PAINT，重画客户区域。

bool PostMessage(uint msg, WPARAM wParam = 0, LPARAM lParam=0) const;

从应用程序中给窗口发一个消息，不等待消息响应的结果而立即返回。

virtual void PerformCreate(int menuOrId);

被 Create 函数调用，形成窗口对象与界面元素之间的对应。

virtual bool PreProcessMsg(MSG& msg);

在分配消息前调用。如果你要重载此函数，切记要调用原函数以保证加速键消息被正确分配。

TWindow * Previous();

返回子窗口表中上一个兄弟窗口的指针。

LRESULT ReceiveMessage(uint msg, WPARAM wParam = 0, LPARAM lParam = 0);

被 StdWndProc 函数调用，ReceiveMessage 是消息接收后第一个被调用的成员函数。

bool RedrawWindow(TRect * update, HRGN hUpdateRgn, uint redrawFlags = DW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE);

重画窗口中指定的区域。redrawFlags 指出窗口重画内容，可以是表 A-6 中所列的多个的 RDW_Xxxx 的组合。

表 A－6

取　　值	意　　义
RDW_ERASE	如果重画窗口，它将接收 WM_ERASEBKGND 消息，如果 RDW_INVALIDATE 没有被选择，此标志无效
RDW_FRAME	如果重画相交的区域，窗口中全部的非客户区将接收 WM_NCPAINT 消息
RDW_INTERNALPAINT	不管是否包括失效区域，窗口将响应 WM_PAINT 消息
RDW_INVALIDATE	重画 hUpdateRgn 和 update 指定的区域。如果它们的值都是 0，就重画整个窗口

表 A－7 是使窗口有效的标志，表 A－8 中是窗口重画标志。

表 A－7

取　　值	意　　义
RDW_NOERASE	窗口拒绝接收 WM_ERASEBKGND 消息
RDW_NOFRAME	窗口拒绝接收 WM_NCPAINT 消息。标志 RDW_VALIDATE 也要被设置
RDW_NOINTERNALPAINT	窗口拒绝接收内部的 WM_PAINT 消息，但不拒绝接收失效区域的 WM_PAINT 消息
RDW_VALIDATE	使 update 和 hUpdateRgn 所在区域有效

表 A－8

取　　值	意　　义
RDW_ERASENOW	函数返回前，特定的窗口接收 WM_NCPAINT 和 WM_ERASEBKGND 事件
RDW_UPDATENOW	函数返回前，特定的窗口接收 WM_NCPAINT、WM_ERASEBKGND 和 WM_PAINT 消息

virtual bool Register()；

登记窗口类。

bool RegisterHotKey(int idHotKey, uint modifiers, uint virtKey)；

在当前程序中登记 ID 值指定的热键。

static void ReleaseCapture()；

释放鼠标器控制权。

HANDLE RemoveProp(uint16 atom) const；

去掉指定的特征表中的特征值。

HANDLE RemoveProp(const char far * str) const；

去掉指定的特征表中的特征值。

void RouteCommandEnable(HWND hInitCmdTarget, TCommandEnabler& ce)；

把初始目标窗口对象和当前对象连接起来。

void ScreenToClient(TPoint& point) const;

将一点的屏幕坐标转化为客户区域坐标。

void ScrollWindow(int dx, int dy, const TRect * scroll = 0, const TRect far * clip = 0);

按照参数值横向或纵向滚动窗口。

void ScrollWindowEx(int dx, int dy, const TRect far * scroll = 0, const TRect far * clip = 0, HRGN hUpdateRgn = 0, TRect far * update = 0, uint flags = 0);

按照参数值横向或纵向滚动窗口，flags 参数决定窗口中的子窗口如何滚动(见表 A-9)。

表 A-9

取　值	意　义
SW_ERASE	给 SW_INVALIDATE 指定的窗口发出一个背景消息后插除失效区域
SW_INVALIDATE	设置 hUpdate 指定的区域为失效区域
SW_SCROLLCHILDREN	滚动所有的子窗口

LRESULT SendDlgItemMessage(int childId, uint msg, WPARAM wParam = 0, LPARAM lParam = 0);

发出一个消息到指定的控件对象。

LRESULT SendMessage(uint msg, WPARAM wParam = 0, LPARAM lParam = 0);

发出一个消息到特定的窗口对象，并等待响应。

void SendNotification(int id, int notifyCode, HWND hCtl, uint msg = WM_COMMAND);

发出一个命令消息，使指定的子窗口发出一个消息到父窗口。

HWND SetActiveWindow();

设置窗口为活动窗口，返回原活动窗口的句柄。

void SetBkgndColor(uint32 color);

设置背景颜色。

void SetCaption(const char far * title);

按照 titile 参数所指的字符串设置窗口的标题。

HWND SetCapture();

获得鼠标器控制权。

static void SetCaretBlinkTime(uint16 milliSecs);

设置插字符闪烁的时间。

static void SetCaretPos(int x, int y);

设置插字符的客户区域的位置坐标。

static void SetCaretPos(const TPoint& pos);

设置插字符的客户区域的位置坐标。

long SetClassLong(int index) const;

设置关于窗口类的 32 位信息。

uint16 SetClassWord(int index, uint16 newWord);

设置关于窗口风格的 16 位信息。

bool SetCursor(TModule * module, TResId resId);

按照模块参数和 ID 值设置光标。

void SetDlgItemInt(int childId, uint value, bool isSigned = true) const;

设置 ID 值指定的控件对象。

void SetDlgItemText(int childId, const char far * text) const;

设置得 ID 值指定的控件对象的文本。

virtual bool SetDocTitle(const char far * docname, int index);

设置 index 所指的文件标题。

void SetFlag(TWindowFlag mask);

设置窗口类标志。

HWND SetFocus();

设置当前焦点窗口。

bool SetMenu(HMENU hMenu);

设置指定的菜单。

void SetModule(TModule * module);

设置缺省的模块。

void SetNext(TWindow * next);

设置兄弟窗口表中下一个窗口的指针。

virtual void SetParent(TWindow * newParent);

设置父窗口指针。

bool SetProp(uint16 atom, HANDLE data) const;

设置窗口性能表的句柄。

bool SetProp(const char far * str, HANDLE data) const;

设置窗口性能表的句柄。

void SetRedraw(bool redraw);

发出 WM_SETREDRAW 消息，设置窗口改变时可以重画(redraw = true)，或不能重画(redraw = false)。

void SetScrollRange(int bar, int minPos, int maxPos, bool redraw = true);

设置滚动条的取值范围。

HWND SetSysModalWindow();

设置系统模块窗口的句柄。

uint SetTimer(uint timerId, uint timeout, TIMERPROC proc = 0);

创建一个与窗口相关的计时器。timerID 设置计时器的 ID 值;timeout 表示时间间隔是多少毫秒;proc 指出响应这个计时器的时间事件的函数地址。

void SetTransferBuffer(void * transferBuffer);

设置数据传送缓冲区。

void SetWindowFont(HFONT font, bool redraw);

为窗口设置特定的字体。

long SetWindowLong(int index, long newLong);

根据 index 值设置关于窗口的信息，表 A-10 列出了 index 的取值和意义。

表 A-10

取 值	意 义
GWL_EXSTYLE	扩展窗口风格
GWL_STYLE	窗口风格
GWL_WNDPROC	窗口回调函数地址
DWL_DLGPROC	对话框回调函数地址 （对话框特有）
DWL_MSGRESULT	对话框信息的取值 （对话框特有）
DWL_USER	附加信息 （对话框特有）

bool SetWindowPlacement(const WINDOWPLACEMENT * place);

设置显示和放置的信息。

void SetWindowPos(HWND hWndInsertAfter, const TRect& rect, uint flags);

根据 TRect& rect 所给区域设置窗口的位置和大小。

void SetWindowPos(HWND hWndInsertAfter, int x, int y, int w, int h, uint flags);

根据 int x, int y, int w, int h 所给区域设置窗口的位置和大小，flags 的值与意义如表 A-11 所列。

表 A-11

取 值	意 义
SWP_DRAWFRAME	给窗口画轮廓
SWP_FRAMECHANGED	发出一个重新计算窗口大小的消息
SWP_HIDEWINDOW	隐藏窗口
SWP_NOACTIVATE	不把窗口设置为活动的
SWP_NOCOPYBITS	放弃窗口中的全部客户区域
SWP_NOMOVE	记忆窗口当前的位置
SWP_NOSIZE	记忆窗口当前的大小
SWP_NOREDRAW	不重画任何窗口变化
SWP_NOZORDER	记忆窗口当前的堆顺序
SWP_SHOWWINDOW	显示窗口

void SetWindowText(const char far * str);

设置窗口标题字符串。

uint16 SetWindowWord(int index, uint16 newWord);

根据 index 值设置关于窗口的信息，表 A-12 是 index 的取值和意义。

表 A-12

取　　值	意　　义
GWW_HINSTANCE	拥有窗口的模块实例指针
GWW_HWNDPARENT	父窗口句柄
GWW_ID	子窗口的 ID 数

void Show(int cmdShow);

确定窗口界面元素有了合法的值后，调用此函数将其显示在屏幕上。cmdShow 的取值和意义如表 A-13 所列。

表 A-13

取　　值	意　　义
SW_SHOWDEFAULT	以缺省的方式显示窗口
SW_HIDE	隐藏窗口并激活其它窗口
SW_MINIMIZE	最小化窗口并使它显示在最前面
SW_RESTORE	与 SW_SHOWNORMAL 相同
SW_SHOW	以当前窗口的大小和位置显示窗口
SW_SHOWMAXIMIZED	以最大化显示窗口
SW_SHOWMINIMIZED	以图标显示窗口
SW_SHOWNA	以当前的方式显示窗口
SW_SHOWMINNOACTIVE	以图标形式形式窗口
SW_SHOWNORMAL	以窗口定义的大小和位置显示窗口
SW_SHOWSMOOTH	在位图更新后显示窗口

void ShowCaret();

以特定的方式在活动窗口的特定位置显示插字符。

void ShowOwnedPopups(bool show);

根据 show 的值显示或隐藏所拥有的弹出窗口。

void ShowScrollBar(int bar, bool show = true);

显示窗口的滚动条。

virtual bool ShowWindow(int cmdShow);

根据 cmdShow 的值显示窗口。cmdShow 的取值和意义如表 A-14 所示。

void ShutDownWindow(int retVal = 0);

关闭窗口。

static void ShutDownWindow(TWindow * win, int retVal = 0);

关闭窗口。

void SubclassWindowFunction();

安装实例转换程序，将旧的窗口函数存入 DefaultProc 中。

表 A - 14

取　　值	意　　义
SW_SHOWDEFAULT	以缺省的方式显示窗口
SW_HIDE	隐藏窗口并激活其它窗口
SW_MINIMIZE	最小化窗口并使它显示在最前面
SW_RESTORE	与 SW_SHOWNORMAL 相同
SW_SHOW	以当前窗口的大小和位置显示窗口
SW_SHOWMAXIMIZED	以最大化显示窗口
SW_SHOWMINIMIZED	以图标显示窗口
SW_SHOWNA	以当前的方式显示窗口
SW_SHOWMINNOACTIVE	以图标形式形式窗口
SW_SHOWNORMAL	以窗口定义的大小和位置显示窗口
SW_SHOWSMOOTH	在位图更新后显示窗口

virtual uint Transfer(void * buffer, TTransferDirection direction);

在窗口和数据缓冲区中传送数据。

virtual void TransferData(TTransferDirection direction);

窗口对象创建和取消时对应此函数。

bool UnregisterHotKey(int idHotKey);

撤销指定热键的登记记录。

void UpdateWindow();

通过发出 WM_PAINT 消息，更新窗口的客户区域。

void Validate();

确认窗口客户区域，用来绘图。

void ValidateRect(const TRect& rect);

确认 TRect& rect 指定的区域，用来绘图。

void ValidateRgn(HRGN hRgn);

确认 HRGN hRgn 指定的区域，用来绘图。

static HWND WindowFromPoint(const TPoint& point) const;

返回指定一点所在窗口的句柄。

virtual LRESULT WindowProc(uint msg, WPARAM wParam, LPARAM lParam);

通过调用 EvCommand 来响应 WM_COMMAND 消息；通过调用 EvCommandEnable 来响应 WM_COMMAND_ENABLE 消息；分配其它消息。

bool WinHelp(const char far * helpFile, uint command, uint32 data);

激活系统的帮助程序。helpFile 指向要显示的帮助文件；command 指出帮助请求的类型，包括 HELP_CONTEXT、HELP_HELPONHELP、HELP_INDEX、HELP_MULTIKEY、HELP_QUIT 或 HELP_SETINDEX。data 包括了帮助标题关键词。

Protected 数据成员：

uint32 BkgndColor;

保存当前的背景颜色。

TModule * CursorModule;

保存当前的光标模块 ID 值。

TResId CursorResId;

保存当前光标资源 ID 值。

HACCEL hAccel;

保存与窗口有关的加速键表的句柄。

HCURSOR HCursor;

保存当前的光标句柄。

void * TransferBuffer;

指向一个数据传送缓冲区的指针。

Protected 成员函数:

virtual void CleanupWindow();

在窗口对象指针非法前调用。

void DispatchScroll(uint scrollCode, uint thumbPos, HWND hWndCtrl);

被 EvHScroll 和 EvVScroll 函数调用来分配滚动条消息。

virtual char far * GetClassName();

返回窗口登记的类名。

virtual void GetWindowClass(WNDCLASS& wndClass);

用属性填充指定的 Windows 注册类。

void Init(TWindow * parent, const char far * title, TModule * module);

初始化窗口对象。

void LoadAcceleratorTable();

加载加速键表。

void RemoveChild(TWindow * child);

清除指定的子窗口。

virtual void SetupWindow();

窗口对象创建之后所调用的第一个成员函数。

事件响应表：事件响应表共有两个：一个是标准的事件响应表(表 A－15)，一个是 Win32 程序事件响应表(表A－16)。

表 A－15

EV_ macros	成员函数
EV_WM_CREATE	EvCreate
EV_WM_CLOSE	EvClose
EV_WM_DESTROY	EvDestroy
EV_WM_SIZE	EvSize
EV_WM_MOVE	EvMove

续表

EV_ macros	成员函数
EV_WM_NCDESTROY	EvNcDestroy
EV_WM_QUERYENDSESSION	EvQueryEndSession
EV_WM_COMPAREITEM	EvCompareItem
EV_WM_DELETEITEM	EvDeleteItem
EV_WM_DRAWITEM	EvDrawItem
EV_WM_MEASUREITEM	EvMeasureItem
EV_WM_CHILDINVALID	EvChildInvalid
EV_WM_VSCROLL	EvVScroll
EV_WM_HSCROLL	EvHScroll
EV_WM_PAINT	EvPaint
EV_WM_SETCURSOR	EvSetCursor
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_COMMAND(CM_EXIT, CmExit)	CmExit
EV_WM_SYSCOLORCHANGE	EvSysColorChange
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_ERASEBKGND	EvEraseBkgnd

表 A-16

EV_ macros	成员函数
EV_MESSAGE(WM_CTLCOLORMSGBOX, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLOREDIT, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLORLISTBOX, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLORBTN, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLORDLG, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLORSCROLLBAR, EvWin32CtlColor)	EvWin32CtlColor
EV_MESSAGE(WM_CTLCOLORSTATIC, EvWin32CtlColor)	EvWin32CtlColor

A.2　TWindowAttr 结构

头文件：window.h

描述：一个典型的 Windows 应用程序具有许多不同风格的窗口对象，不同的窗口风格是通过设置窗口属性参数 TWindowAttr Attr 确定的，窗口风格和窗口标题都是在窗口初始化是设置的，创建窗口元素时将

使用这些参数。

Public 数据成员：

TResId AccelTable；

窗口的加速键表 ID 值.。

uint32 ExStyle；

扩展风格，其值及意义如表 A-17 所示。

表 A-17

ExStyle 的值	意　　义
WS_EX_ACCEPTFILES	允许窗口的拖放功能
WS_EX_DLGMODALFRAME	创建一个具有双边界的窗口
WS_EX_NOPARENTNOTIFY	创建一个子窗口，当它创建或删除时不向父窗口发出消息
WS_EX_TOPMOST	创建一个总是浮在最上面的窗口，即使它是非活动窗口，也浮在最上面
WS_EX_TRANSPARENT	创建一个透明的窗口，它下面的窗口不会被它所覆盖

int Id；

子窗口的 ID 值。

TResId Menu；

窗口菜单的 ID 值。

char far * Param；

TMDIClient 用于保存多文档界面信息。

uint32 Style；

窗口风格参数，其值及意义如表 A-18 所示。

表 A-18

style 的值	意　　义
WS_BORDER	创建一个带边界的窗口
WS_CAPTION	创建一个带标题的窗口
WS_CHILD	创建一个子窗口，不能与 WS_POPUP 一起使用
WS_CHILDWINDOW	创建一个子窗口
WS_CLIPCHILDREN	当在父窗口中绘图时，不包括子窗口所占的区域
WS_CLIPSIBLINGS	裁剪彼此相关的子窗口
WS_DISABLED	创建一个禁止使用的窗口
WS_DLGFRAME	创建一个带双边界无标题窗口
WS_GROUP	定义可以用方向键在彼此之间移动的一组控件中的第一个控件
WS_HSCROLL	创建一个具有横向滚动条的窗口

续表

style 的值	意　　义
WS_MAXIMIZE	创建一个最大化窗口
WS_MAXIMIZEBOX	创建一个具有最大化按钮的窗口
WS_MINIMIZE	创建一个最小化窗口
WS_MINIMIZEBOX	创建一个具有最小化按钮的窗口
WS_OVERLAPPED	创建一个具有标题和边界的覆盖窗口
WS_OVERLAPPEDWINDOW	创建一个覆盖窗口带有风格：WS_OVERLAPPED、WS_CAPTION、WS_SYSMENU、WS_THICKFRAME、WS_MINIMIZEBOX 和 WS_MAXIMIZEBOX
WS_POPUP	创建一个弹出窗口。不能与 WS_CHILD 一起使用
WS_POPUPWINDOW	创建一个弹出窗口带有风格：WS_BORDER、WS_POPUP 和 WS_SYSMENU。WS_CAPTION 和 WS_POPUPWINDOW 风格连用创建一个系统菜单
WS_SYSMENU	创建一个标题条上有系统菜单的窗口。必须有 WS_CAPTION 风格
WS_TABSTOP	指定用户可以用 TAB 键选择移动的控件
WS_THICKFRAME	创建一个粗边界的窗口，可以改变窗口大小
WS_VISIBLE	创建一个可见的覆盖或弹出窗口
WS_VSCROLL	创建一个具有纵向滚动条的窗口

int X, Y, W, H;

窗口左上角在屏幕的横纵坐标；窗口的初始化宽度和高度。

类型定义：enum TWindowFlag

A.3　TApplication 类

头文件：applicat.h

描述：TApplication 类封装了初始化、运行时管理和关闭 Windows 应用程序功能。TApplication 类也包括了参数 hPrevInstance 和 nCmdShow 的值，这与非 ObjectWindows 应用程序中的 WinMain 的参数是等同的。因为 TApplication 类是派生于 TModule 的，所以它也拥有 TModule 类中的所有函数。TApplication 在 OWL 中的位置如图 A－2 所示。

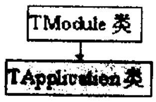


图 A－2　TApplication 类在 OWL 5.0 中的位置

Public 数据成员：

HACCEL HAccTable;

当前应用程序加速键表的句柄，该句柄向后兼容。对于新的应用程序，你可以使用 TWindowAttr::AccelTable 为每一个窗口提供加速键表。

HINSTANCE hPrevInstance;

Windows 应用程序的上一个运行实例句柄。如果 hPrevInstance 是 0，说明这个应用程序实例之前没有该应用程序的其它实例在运行。在 Win32 中，hPrevInstance 总是 0。

int nCmdShow;

应用程序主窗口的初始显式方式(可以是最大化或是图标形式显示)，与 WinMain 中的 nCmdShow 参数有相同含义。nCmdShow 可以取的值和它们的意义见表 A-19。

表 A-19

取　　值	意义解释
SW_SHOWDEFAULT	以缺省的方式显示
SW_HIDE	隐藏窗口
SW_MINIMIZE	最小化特定窗口
SW_SHOW	以当前设置的位置和大小激活窗口
SW_SHOWMAXIMIZED	最大化显示
SW_SHOWMINIMIZED	最小化显示
SW_SHOWNA	以当前状态显示窗口
SW_SHOWNOACTIVATE	以图标方式显示
SW_SHOWNORMAL	以原始的位置和大小显示
SW_SHOWSMOOTH	更新位图后显示窗口

Public 构造函数和析构函数：

```
TApplication(const char far * name = 0,
             TModule * & gModule = ::Module,
             TAppDictionary * appDict = 0);
TApplication(const char far * name,
             HINSTANCE hInstance,
             HINSTANCE hPrevInstance,
             const char far * cmdLine,
             int cmdShow,
             TModule * & gModule = ::Module,
             TAppDictionary * appDict = 0);
~TApplication();
```

Public 成员函数：

int BeginModal(TWindow * window, int flags = MB_APPLMODAL);

用此函数开始窗口消息循环。先冻结窗口，进行消息循环，完成后再解冻窗口。flags 标志的取值决定了 BeginModal 如何处理窗口，其关系如表 A-20 所示。

表 A-20

取　　值	意义解释
MB_APPLMODAL	冻结 window 参数指定的窗口，如果 window 为 0，就不冻结窗口
MB_SYSTEMMODAL	窗口成为窗口设置的系统模式
MB_TASKMODAL	最上层的窗口被冻结

bool BWCCEnabled() const;

返回是否开启了 BWCC(Borland Custom Controls library)，如果 BWCC 开启，返回 true;否则返回 false。

virtual bool CanClose();

如果选择 OK 则返回 true。缺省时 CanClose 调用主窗口对象的成员函数 CanClose，当主窗口和文件管理器都返回 true 时才返回 true，否则返回 false。这个成员函数很少被重载，通常是通过修改主窗口的 CanClose 成员函数来调节应用程序关闭时的动作。

void Condemn(TWindow * win);

清除窗口。

bool Ctl3dEnabled()const;

返回是否开启了 Ctl3D，如果 Ctl3D 开启了，返回 true;否则返回 false。

void EnableBWCC(bool enable = true, uint Language = 0);

装载 BWCC.DLL(如果你运行 16 位应用程序)或 BWCC32.DLL(如果你运行 32 位应用程序)。缺省时，BWCC 是开启的。

void EnableCtl3d(bool enable = true);

装载 CTL3D DLL，要关闭 CTL3D DLL 就把 enable 设置成 false。

void EnableCtl3dAutosubclass(bool enable);

当 CTL3D 开启时，开启或关闭 CTL3D 的自动转换功能。如果开启此功能，那么任何一个非 ObjectWindows 对话框都将具有一个 3-D 的界面。

void EndModal(int result);

调用此函数来关闭模式窗口的模式消息循环。如果出现错误，将 result 设置为-1。

bool Find(TEventInfo &, TEqualOperator = 0);

因为 TApplication 类本身没有事件处理表，故 Find 函数将事件转移到文件管理器中处理。

TModule * GetBWCCModule() const;

返回 BWCC 的模式指针。

TModule * GetCtl3dModule() const;

返回 CTL3D 的模式指针。

TDocManager * GetDocManager();

返回应用程序的文件管理器对象的指针。

TFrameWindow * GetMainWindow();

返回应用程序主窗口的指针。

TWindow * GetWindowPtr(HWND hWnd) const;

返回特定窗口句柄的窗口类指针。

void GetWinMainParams();

初始化一个静态的应用程序对象。

virtual int MessageLoop();

在应用程序生存期间处理消息循环。

void PostDispatchAction();

如果应用程序的消息循环处理函数没有被使用，那么每当消息分配以后就要调用此函数。

virtual void PreProcessMenu(HMENU hmenu);

在窗口菜单显示之前就运行菜单操作。

virtual bool ProcessAppMsg(MSG& msg);

检查所有的消息，并调用虚函数 TWindow::PreProcessMsg 来接收消息。如果你的应用程序中不包含任何特殊过程例如非模式对话框、加速键表以及多文档的加速键表消息，你就可以重载这个函数使它返回 false。

bool PumpWaitingMessages();

被 MessageLoop 函数调用，PumpWaitingMessages 处理和分配所有的等待消息直到消息队列为空。当接收了 WM_QUIT 消息时，它也设置 BreakMessageLoop。

int QueryThrow();

检查是否有例外的挂起，并返回一系列位标志。

void ResumeThrow();

检查并重新丢弃一个例外的挂起。

virtual int Run();

TApplication::Run 的功能是运行一个应用程序，但是在此期间它还完成了一系列的任务。

```
static void SetWinMainParams(HINSTANCE hInstance,
                             HINSTANCE hPrevInstance,
                             const char far * cmdLine,
                             int cmdShow);
```

ObjectWindows 应用程序缺省的 WinMain 函数调用 SetMainWinParams 函数，以便 TApplication 可以存储参数供以后使用。

void SuspendThrow(xalloc& x);

void SuspendThrow(xmsg& x);

void SuspendThrow(TXOwl& x);

void SuspendThrow(int);

以上均为程序例外挂起。

void Uncondemn(TWindow * win);

从废除窗口列中去掉一个窗口。

Protected 成员函数：

virtual bool IdleAction(long idleCount);

当消息队列中没有消息时，ObjectWindows 调用 IdleAction 函数。你可以重载这个函数来做一些后台处理工作。但是，缺省的动作是给主窗口一个处理空闲的机会，并且一直 IdleAction 返回 true。idleCount 代表 IdleAction 函数在消息之间被调用的次数。

virtual void InitApplication();

ObjectWindows 程序调用 InitApplication 函数来初始化第一个实例。对于其它实例，这个函数不被调用。

virtual void InitInstance();

与 InitApplication 不同，每一个应用程序都必须调用这个初始化函数来做一些必要的初始化工作。InitInstance 函数调用 InitMainWindow 函数，然后通过调用 TWindow::Create 和 TWindow::Show 函数来创建和显示主窗口元素。如果不能创建主窗口，就产生一个例外。如果你要重定义这个函数，必须保

证调用原来的 TApplication::InitInstance 函数。

virtual void InitMainWindow();

缺省时，InitMainWindow 函数构造一个通用 TFrameWindow 的窗口对象，窗口的标题就用应用程序的名字。你可以重定义这个函数来构造一个有用的主窗口对象，它可以是 TFrameWindow 类的对象，也可以是 TFrameWindow 类的派生类的对象。主窗口必须是最上层的窗口，即它必须是从 TFrameWindow 类中产生(或派生)的。InitMainWindow 函数可以被重载以建立你自己的窗口。

TFrameWindow * SetDocManager(TDocManager * docManager);

设置一个应用程序的文件管理器的指针。

TFrameWindow * SetMainWindow(TFrameWindow * window);

设置一个新的主窗口，并设置窗口的 WM_MAINWINDOW 标志。

virtual int TermInstance(int status);

处理每一个 ObjectWindows 应用程序运行实例的终止。

A.4　TFrameWindow 类

头文件：framewin.h

描述：TFrameWindow 是从 TWindow 类派生出来的，它控制着窗口的特定行为比如键盘控制和客户窗口的命令执行。TFrameWindow 类是可流类，它在 OWL 5.0 中的位置如图 A-3 所示。

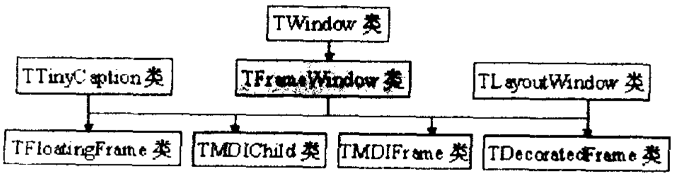


图 A-3　TFrameWindow 类在 OWL 5.0 中的位置

窗口的一系列区域可分为框架窗口和客户窗口两部分，框架窗口的区域包括边界、系统菜单、工具条和状态条，其它就属于客户窗口区域。框架窗口可以支持一个客户窗口，但它与客户窗口保持了相对独立，你可以改变客户窗口而不会影响框架窗口。

在 ObjectWindows 体系中用到这种 frame/client(框架窗口/客户窗口)结构的还有 TMDIChild 类，它们都可以包括一个客户窗口。为用户区域提供一个分类的类可以使你的程序变得更加灵活。比如一个对话框作为客户窗口可以放到其他的框架窗口中。

Public 数据成员：

bool KeyboardHandling;

表明是否接管键盘的标志。

Public 构造函数和析构函数：

```
TFrameWindow(TWindow * Parent,
             const char far * title = 0,
             TWindow * clientWnd = 0,
             bool shrinkToClient = false,
             TModule * module = 0);
```

构造一个窗口对象。parent 参数接收一个父窗口，如果构造的窗口是主窗口，则值为 0；title 参数缺省

为 0，它包括要显示的窗口标题；clientWnd 是框架窗口的客户窗口，如果没有客户窗口，则值为 0。shrinkToClient 指明是否让客户窗口的大小与框架窗口相适应，这个参数只影响主窗口的大小。当框架窗口中含义客户窗口时，设置 shrinktoClient 参数将使客户窗口改变大小与框架窗口相匹配；不设置 shrinktoClient 参数将使框架窗口改变大小与客户窗口相匹配。

TFrameWindow(HWND hWnd, TModule * module = 0);

构造一个窗口对象作为一个非 ObjectWindows 窗口的别名。hWnd 参数是 TFrameWindow 要控制的已存在窗口的句柄；module 传给基类构造函数。

~TFrameWindow();

Public 成员函数：

virtual bool AssignMenu(TResId menuResId);

设置 Attr. Menu 为 menuResId 参数提供的菜单，取消以前的设置。如果 HWindow 不为 0，装入并设置窗口菜单，取消所有已存在的菜单。

void EnableKBHandler();

设置一个标志表明接收者要求管理键盘。

virtual TWindow * GetClientWindow();

返回一个指向客户窗口的指针。如果你想访问 TMDIChild 框架窗口中的一个窗口对象，你就需要用到这个成员函数。

virtual HWND GetCommandTarget();

定位并返回响应 command 和 command enable 消息的子窗口句柄。如果当前应用程序没有焦点或焦点在应用程序的工具条上，GetCommandTarget 返回最近的子窗口。如果要改变 command 消息处理方式，主窗口类可以重载这个函数。TFrameWindow 类的 EvCommand 和 EvCommandEnable 函数调用 GetCommandTarget 来找到响应 command 和 command enable 消息的子窗口。

const TMenuDescr * GetMenuDescr();

返回菜单描述对象指针。

bool HoldFocusHWnd(HWND hWndLose, HWND hWndGain);

重载 TWindow 类的同名的成员函数，响应失去焦点的子窗口获得其句柄的请求，子窗口的句柄存储在 HwndRestoreFocus 中。

void IdleAction(long idleCount);

重载 TWindow 类的同名的成员函数，当没有消息需要处理时 TApplication 类调用子窗口的 IdleAction 函数。TFrameWindow 使用这个空闲时间为菜单条处理 command enable 消息，它也处理每一个子窗口的 IdleAction 函数。IdleAction 可被重载来进行后台处理。

bool MergeMenu(const TMenuDescr&);

将 childMenuDescr 菜单描述中的菜单与窗口自己的菜单描述融合起来，并在窗口中显示得到的菜单。

bool PreProcessMsg(MSG& msg);

重载 TWindow 类的同名的成员函数。此函数预先处理窗口消息。如果子窗口要求控制键盘，PreProcessMsg 函数处理任何加速键消息然后处理其他的键盘消息。

bool RestoreMenu();

恢复窗口的缺省菜单。

virtual TWindow * SetClientWindow(TWindow * clientWnd);

给特定窗口设置客户窗口。

bool SetDocTitle(const char far * docname, int index);

重载 TWindow 类的同名的成员函数，将浏览的数字贴在窗口标题上并将数字显示在屏幕上。如果你

不想要缺省的执行方式(在屏幕上显示数字)你可以重载它，比如你想显示“二”而不是“2”。

bool SetIcon(TModule * iconModule, TResId iconResId);

按照 iconResId 所给的资源 ID 值设置图标，你可以用表 A-21 中所列的缺省图标。

表 A-21

图标值	意义解释
IDI_APPLICATION	应用程序的缺省图标
IDI_ASTERISK	提示信息图标
IDI_EXCLAMATION	叹号警告信息图标
IDI_HAND	手形警告信息图标
IDI_QUESTION	问号请求一个答复

virtual bool SetMenu(HMENU newMenu);

重载 TWindow 类的同名的非虚成员函数。SetMenu 函数以 newMenu 参数设置窗口菜单，如果 newMenu 的值为 0，删除窗口的当前菜单;SetMenu 函数返回 0 说明菜单没有改变，否则返回一个非 0 值。

void SetMenuDescr(const TMenuDescr& menuDescr);

设置新的菜单描述。

Protected 数据成员:

TWindow * ClientWnd;

ClientWnd 指向框架中的用户窗口。

int DocTitleIndex;

为文件标题保存检索号。

HWND HWndRestoreFocus;

保存恢复焦点的子窗口句柄。

TModule * MergeModule;

说明框架窗口中菜单的模式来源。TDecoratedFrame 类用这个数据成员获得显示在窗口底部的菜单提示，认为菜单和它的提示来自相同的地方。

Protected 成员函数:

LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);

提供额外的 commands 消息处理过程并让焦点窗口和它的父窗口首先处理这个 command 消息。

void EvCommandEnable(TCommandEnabler& ce);

处理框架窗口的菜单项的检验状态。EvCommandEnable 使用 TWindow 类的 RouteCommandEnable 成员函数来完成主要的工作。

bool EvEraseBkgnd(HDC);

EvEraseBkgnd 函数在 HDC 中擦除窗口的背景。如果擦除成功返回 true，否则返回 false。

HANDLE EvInitMenuPopup(HMENU hPopupMenu, uint index, bool sysMenu);

EvInitMenuPopup 函数可在弹出菜单显示之前调用，它可以在应用程序显示菜单之前修改菜单项。EvInitMenuPopup 控制着菜单项是否可用，是否为检验状态并且设置菜单项的字符串。HMENU 参数给出菜单的句柄；index 参数是弹出菜单的检索值；sysMenu 参数表明弹出菜单是否是系统菜单。

void EvPaint();

在客户窗口响应 WM_PAINT 消息用来绘制图标化窗口的图标或允许客户窗口绘制图标。

void EvParentNotify(uint event, uint childHandleOrX, uint childIDOrY);

通过响应一个消息来通知父窗口有一个事件发生。如果客户窗口关闭了，就关闭父窗口；如果设置了 shrinkToClient 参数并改变了客户窗口的大小，框架窗口也随之改变。当 TFrameWindow 的客户窗口改变了，TFrameWindow 对象接收 WM_PARENTNOTIFY 消息并给自身发一个改变消息。没有这个消息，空的框架窗口将保留，由客户窗口来决定如何改变框架窗口。如果你不想这样做，你可以通过继承框架窗口类并由你自己的函数来处理 EvParentNotify 或 EvClose 消息。

HANDLE EvQueryDragIcon();

响应一个 WM_QUERYDRAGICON 消息，发出一个准备拖动的图标化窗口。EvQueryDragIcon 函数使用 SetIcon 中设置的图标而不是缺省的图标。

void EvSetFocus(HWND hWndLostFocus);

恢复活动窗口的焦点。hWndLostFocus 包括了失去焦点的窗口句柄。

void EvSize(uint sizeType, TSize& size);

响应 WM_SIZE 消息调节客户窗口的大小以适应客户区域的大小。

void Init(TWindow * clientWnd, bool shrinkToClient);

虚派生类在构造函数完成之前调用这个函数来作初始化工作。

void SetupWindow();

调用 TWindow::SetUpWindow 函数来建立一个子窗口列。SetupWindow 调节客户窗口、装载框架窗口的菜单模式并初始化 HwndRestoreFocus。

事件响应表：见表 A-22。

表 A-22

EV_ macros	成员函数
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_WM_INITMENUPOPUP	EvInitMenuPopup
EV_WM_PAINT	EvPaint
EV_WM_PARENTNOTIFY	EvParentNotify
EV_WM_QUERYDRAGICON	EvQueryDragIcon (Windows only)
EV_WM_SETFOCUS	EvSetFocus
EV_WM_SIZE	EvSize

A.5 TControl 类

头文件：control.h

描述：TControl 类是一系列控件类的基类，包括 TScrollBar，TControlGadget 和 TButton 等。派生控件类的对象通常用来描述控件的界面元素，一个控件对象负责在父窗口中创建一个控件界面元素。在对话框中，控件对象可以将你的应用程序和对话框中的控件联系起来。TControl 类是可流类，它在 OWL 5.0 中所位置如图 A-4 所示。

Public 构造函数:

```
TControl(TWindow * parent, int id,
         const char far * title,
         int x, int y, int w, int h,
         TModule * module = 0);
TControl(TWindow * parent,
         int resourceId,
         TModule * module = 0);
```

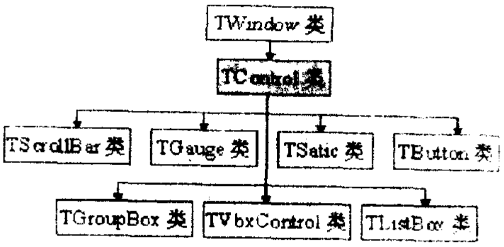


图 A-4 TControl 类在 OWL 5.0 中的位置

Protected 成员函数:

virtual int CompareItem (COMPAREITEMSTRUCT far& compareInfo);

CompareItem 函数比较两个项目，也用于自绘按钮和列表框控件。派生类将提供比较的具体逻辑。

virtual void DeleteItem (DELETEITEMSTRUCT far& deleteInfo);

DeleteItem 函数删除一个项目，用于自绘按钮和列表框控件时，由派生类将提供删除的具体逻辑。

virtual void DrawItem(DRAWITEMSTRUCT far& drawInfo);

DrawItem 函数响应一个消息，当控件需要重画时，TWindow 给可绘制的控件发出这个消息。如果全部的控件需要重画时，TControl::DrawItem 调用 ODADrawEntire;如果控件的选择状态改变，就调用 ODASelect;如果焦点进入或移出控件时，调用 ODAFocus 。

LRESULT EvCompareItem(uint ctrlId, COMPAREITEMSTRUCT far& comp);

为自绘按钮处理一个 WM_COMPAREITEM 消息。

virtual void EvDeleteItem(uint ctrlId, DELETEITEMSTRUCT far& deleteInfo);

为自绘按钮处理一个 WM_DELETEITEM 消息。

void EvDrawItem(uint ctrlId, DRAWITEMSTRUCT far& draw);

响应一个 WM_DRAWITEM 消息。

void EvMeasureItem(uint ctrlId, MEASUREITEMSTRUCT far& meas);

响应一个 WM_MEASUREITEM 消息。

void EvPaint();

如果控件是一个预定义类，EvPaint 调用 TWindow::DefaultProcesing 类绘制控件；否则，它将调用 TWindow::EvPaint 来绘制。

virtual void MeasureItem (MEASUREITEMSTRUCT far& measureInfo);

处理一个 WM_MEASUREITEM 消息。

virtual void ODADrawEntire(DRAWITEMSTRUCT far& drawInfo);

当需要控件为可绘状态时，响应一个发给可绘制控件的广播事件。在可绘制控件类中 ODADrawEntire 函数可以被重定义，用来进行特殊的绘制。

virtual void ODAFocus(DRAWITEMSTRUCT far& drawInfo);

当焦点进入或移出时响应一个发给可绘制控件的广播事件。在可绘制控件类中，ODAFocus 函数可以重定义，以用来进行特殊的绘制。

virtual void ODASelect(DRAWITEMSTRUCT far& drawInfo);

当控件的选择状态改变时响应一个发给可绘制控件的广播事件。在可绘制控件类中 ODASelect 函数可以重定义，以用来进行特殊的绘制。

事件响应表：见表 A－23。

表 A－23

EV_ macros	成员函数
EV_WM_PAINT	EvPaint
EV_WM_COMPAREITEM	EvCompareItem
EV_WM_DELETEITEM	EvDeleteItem
EV_WM_DRAWITEM	EvDrawItem
EV_WM_MEASUREITEM	EvMeasureItem

A.6　TScrollBar 类

头文件：scrollba.h

描述：TScrollBar 对象描述了横向和纵向的滚动条控件。大多数的 TScrollBar 成员函数管理滚动条的指示位置的小按钮，控制它的位置和范围。TScrollBar 类的一个比较特殊的地方是它的一系列成员函数是通过响应广播消息自动的调节位置按钮的。所以不要在具有窗口滚动条的窗口中使用 TScrollBar。

TScrollBar 类是可流类，它在 OWL 5.0 中的位置如图 A－5 所示。

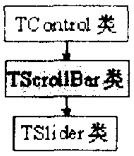


图 A－5　TScrollBar 类在 OWL 5.0 中的位置

Public 数据成员：

int LineMagnitude;

LineMagnitude 的值是滚动条滚动的最小单位值，即滚动条以最小的步子滚动一下所走的距离。TScrollBar 的构造函数设置 LineMagnitude 的缺省值为 1(滚动范围的缺省值为 0～100)。

int PageMagnitude;

PageMagnitude 的值是滚动条滚动的最大单位值，即用户在滚动条的滚动区域按下鼠标使滚动条以最大的步子滚动一下所走的距离。TScrollBar 的构造函数设置 PageMagnitude 的缺省值为 10(滚动范围的缺省值为 0～100)。

Public 构造函数：

```
TScrollBar(TWindow * parent,
           int id,
           int x, int y, int w, int h,
           bool isHScrollBar,
           TModule * module = 0);
TScrollBar(TWindow * parent,
           int resourceId,
           TModule * module = 0);
```

Public 成员函数：

virtual int DeltaPos(int delta);

按照 delta 的值调用 SetPosition 函数来改变滚动条指示按钮的位置，返回新的指示按钮位置。delta 值为正时指示按钮向右或向下；delta 值为负时指示按钮向左或向上。

void EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl);

响应 TWindow::DispatchScroll 发出的消息。

void EvVScroll(uint scrollCode, uint thumbPos, HWND hWndCtl);

响应 TWindow::DispatchScroll 发出的消息。

virtual int GetPosition() const;

返回位置按钮当前的位置。

virtual void GetRange(int& min, int& max) const;

返回滚动条当前滚动范围的最大值和最小值。

virtual void SBBottom();

调用 SetPosition 函数将位置指示按钮移到滚动条的最下面或最右面。SBBottom 用来响应当位置指示按钮被拖到滚动条最下面或最右面的事件。

virtual void SBLineDown();

调用 SetPosition 函数将位置指示按钮向下面或向右面滚动（根据 LineMagnitude 设置的单位）。SBLineDown 用来响应鼠标在向下面箭头或向右面箭头按下的事件。

virtual void SBLineUp();

调用 SetPosition 函数将位置指示按钮向上面或向左面滚动（根据 LineMagnitude 设置的单位）。SBLineUp 用来响应鼠标在向上面箭头或向左面箭头按下的事件。

virtual void SBPageDown();

调用 SetPosition 函数将位置指示按钮向下面或向右面滚动（根据 PageMagnitude 设置的单位）。SBLineDown 用来响应鼠标在滚动条的滚动区域按下的事件。

virtual void SBPageUp();

调用 SetPosition 函数将位置指示按钮向上面或向左面滚动（根据 PageMagnitude 设置的单位）。SBPageUp 用来响应鼠标在滚动条的滚动区域按下的事件。

virtual void SBThumbPosition(int thumbPos);

调用 SetPosition 函数移动位置指示按钮。SBThumbPosition 响应当位置指示按钮设置事件。

virtual void SBThumbTrack(int thumbPos);

调用 SetPosition 函数移动位置指示按钮。SBThumbTrack 响应当位置指示按钮拖动事件。

virtual void SBTop();

调用 SetPosition 函数将位置指示按钮移到滚动条的最上面或最左面。SBTop 用来响应当位置指示按钮被拖到滚动条最上面或最左面的事件。

virtual void SetPosition(int thumbPos);

移动位置指示按钮到 ThumbPos 指定的位置。如果 ThumbPos 超出了滚动条的滚动范围，位置指示按钮将移动到在范围内最接近的位置。

virtual void SetRange(int min, int max);

根据 min 和 max 的值设置滚动条的范围。

uint Transfer(void * buffer, TTransferDirection direction);

从 buffer 所指的缓冲区传入或传出滚动条数据。buffer 指向一个 TScrollBarData 结构的指针。数据是从缓冲区传入还是传出到缓冲区是由 direction 参数决定的，如果 direction 是 tdGetData，数据从缓冲区中传出；如果 direction 是 tdSetData，数据从缓冲区中传入；Transfer 总是返回传输数据的大小(TScrollBar-

Data 结构的大小)。如果只想得到这个结构的大小而不想传入或传出数据，就设置 direction 的值为 td-SizeData。

Protected 成员函数：

char far * GetClassName();

返回滚动条的注册类名"SCROLLBAR"。

void SetupWindow();

设置滚动条的滚动范围为 0～100，想要改变滚动范围就调用函数 SetRange。

事件响应表：见表 A-24。

表 A-24

EV_ macros	成员函数
EV_WM_HSCROLL	EvHScroll
EV_WM_VSCROLL	EvVScroll

A.7　TScrollBarData 结构

头文件：scrollba.h

描述：TScrollBarData 结构包含了表示滚动条上位置指示按钮的范围的整数值。TScrollBar 类的成员函数 GetRange 调用 TScrollBarData 来获得在滚动条中位置指示按钮的最大值和最小值。GetPosition 调用 TScrollBarData 来获得滚动条中当前位置指示按钮的位置。

Public 数据成员：

int HighValue;

滚动条中位置指示按钮的最大值。

int LowValue;

滚动条中位置指示按钮的最小值。

int Position;

滚动条中当前位置指示按钮的位置。

A.8　TSlider 类

头文件：slider.h

描述：作为 TScrollBar 的派生类，TSlider 定义了 slider 控件的基本行为。与普通的滚动条一样，slider 具有最大和最小的位置，可以通过鼠标和键盘来移动指示指针、键盘的 Home 和 End 键使指示指针移动到最大和最小的位置上。可以通过 TSlider 的成员函数来设置指示指针的位置，设置指针移动一步的距离。TSlider 类在 OWL 5.0 中的位置如图 A-6 所示。

Public 构造函数和析构函数：

```
TSlider(TWindow * parent,
        int id,
        int X, int Y, int W, int H,
        TResId thumbResId,
        TModule * module = 0);
~TSlider();
```

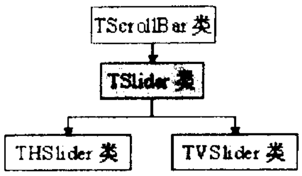


图 A－6　Slider 类在 OWL 5.0 中的位置

Public 成员函数：

int GetPosition() const;

重载 TScrollBar 的成员函数，返回的指示指针当前位置。

void GetRange(int &min, int &max) const;

重载 TScrollBar 的成员函数，返回 slider 的指示范围。

void SetPosition(int thumbPos);

重载 TScrollBar 的成员函数，将指示指针移动到 thumbPos 所指的位置上。如果 thumbPos 超出了 slider 的范围，指示指针就移动到在范围中最接近的位置。

void SetRange(int min, int max);

重载 TScrollBar 的成员函数，设置 slider 的指示范围。

void SetRuler(int ticGap, bool snap = false);

设置 slider 的标尺。每一个 slider 控件都有一个内建的标尺，上面刻有刻度。

Protected 成员函数：

bool EvEraseBkgnd(HDC hDC);

响应 WM_ERASEBKGND 消息，擦除 slider 的背景。

uint EvGetDlgCode(MSG far * msg);

响应 WM_GETDLGCODE 消息，将方向按键和 TAB 键的控制权交给对话框管理者。

void EvKeyDown(uint key, uint repeatCount, uint flags);

将虚键转换为实际动作，移动指示指针。key 保存了虚键代码，repeatCount 保存了同一虚键的重复次数，表 A－25 列出了 flags 的值和这些值对应的虚键代码。

表 A－25

flags 的值	虚键代码
SB_PAGEUP	VK_PRIOR
SB_PAGEDOWN	VK_NEXT
SB_BOTTOM	VK_END
SB_TOP	VK_HOME
SB_LINEUP	VK_LEFT(与 SB_LINELEFT 相同)
SB_LINEUP	VK_UP
SB_LINEDOWN	VK_RIGHT(与 SB_LINERIGHT 相同)
SB_LINEDOWN	VK_DOWN

void EvKillFocus(HWND hWndGetFocus);

响应 WM_KILLFOCUS 消息，窗口失去焦点。

void EvLButtonDblClk(uint modKeys, TPoint& point);

响应 WM_LBUTTONDBLCLK 消息。

void EvLButtonDown(uint modKeys, TPoint& point);

响应鼠标的按键，设置指示指针的位置并开始一个鼠标拖动动作。如果鼠标是在指示指针上按下，EvLButtonDown 就加入滑动状态；如果鼠标是在指针轨道上按下，EvLButtonDown 就执行上下翻页动作；如果鼠标是在标尺上按下，EvLButtonDown 就把指示指针移动到那个位置上。

void EvLButtonUp(uint modKeys, TPoint& point);

如果鼠标按键释放了，EvLButtonUp 函数就结束滚动、翻页或是在标尺上的跳动。

void EvMouseMove(uint modKeys, TPoint& point);

响应鼠标移动的事件，如果是在拖动，EvMouseMove 函数就将指示指针移动到鼠标的位置上。

void EvPaint();

绘制 slider 控件，边框内建标尺、轨道和指示指针。

void EvSetFocus(HWND hWndLostFocus);

响应 slider 得到焦点的事件。

void EvSize(uint sizeType, TSize& size);

响应窗口改变大小的事件。

void GetBkColor(TDC& dc);

发出 WM_CTLCOLOR 消息获得背景颜色。

virtual int HitTest(TPoint& point) = 0;

获得所给坐标在 slider 中的对应位置。

virtual void NotifyParent(int scrollCode, int pos=0) = 0;

发出 WS_HSCROLL 和 WS_VSCROLL 消息到父窗口。

virtual void PaintRuler(TDC& dc) = 0;

绘制内建标尺。

virtual void PaintSlot(TDC& dc) = 0;

绘制轨道。

virtual void PaintThumb(TDC& dc);

绘制指示指针。

virtual int PointToPos(TPoint& point) = 0;

将所给的 X，Y 坐标转换成 slider 的值。

virtual TPoint PosToPoint(int pos) = 0;

将 slider 的值转换成坐标。

virtual void SetupThumbRgn();

设置指示指针的形状。

void SetupWindow();

调用 TScrollBar::SetupWindow 和 SetupThumbRgn 函数设置窗口。

virtual void SlideThumb(TDC& dc, int thumbPos);

将指示指针移动到指定的位置。

int SnapPos(int pos);将 pos 的值转换为满足 slider 范围和刻度的值。

Protected 数据成员：

TColor BkColor；

保存背景颜色。

TRect CaretRect；

保存插字符的区域。

int Max；

保存 slider 范围的最大值。

int Min；

保存 slider 范围的最小值。

static TSize MouseOffset；

显示鼠标补偿。

int Pos；

保存指示指针在 slider 的当前位置。

uint Range；

保存 slider 的范围。

static TDC * SlideDC；

当鼠标按键按下并且指示指针滑动时使用这个参数。

bool Sliding；

保存 slider 控件的指示指针是否在滑动。

int SlotThick；

设置 slider 控件的轨道宽度，缺省时为 17。

bool Snap；

如果 snap 开启则值为 true，否则为 false。

TRect ThumbRect；

保存指示指针的区域。

TResId ThumbResId；

保存指示指针位图的 ID 值。

TRegion * ThumbRgn；

定义指示指针的形状。

int TicGap；

保存指示指针移动一步的距离。

事件响应表：见表 A－26

表 A－26

EV_ macros	成员函数
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_WM_GETDLGCODE	EvGetDlgCode
EV_WM_KEYDOWN	EvKeyDown
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_LBUTTONDBLCLK	EvLButtonDblClk

续表

EV_ macros	成员函数
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_PAINT	EvPaint
EV_WM_SETFOCUS	EvSetFocus
EV_WM_SIZE	EvSize

A.9 THSlider 类

头文件：slider.h

描述：作为 TSlider 类的派生类，THSlider 提供了横向 slider 控件的具体特性，它在 OWL 5.0 中的位置如图A-7所示。

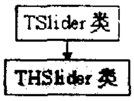


图 A-7 THSlider 类在 OWL 5.0 中的位置

Public 构造函数：

```
THSlider(TWindow * parent,
         int id,
         int X, int Y, int W, int H,
         TResId thumbResId = IDB_HSLIDERTHUMB,
         TModule * module = 0);
```

Protected 成员函数：

int HitTest(TPoint& point);

重载 TSlider 的成员函数，获得指定的坐标在 slider 控件上的对应值。

void NotifyParent(int scrollCode, int pos=0);

重载 TSlider 的成员函数，给父窗口发出 WS_HSCROLL 消息。

void PaintRuler(TDC&);

重载 TSlider 的成员函数，绘制内建标尺。

void PaintSlot(TDC&);

重载 TSlider 的成员函数，绘制轨道。

int PointToPos(TPoint& point);

重载 TSlider 的成员函数，将坐标值转换为指示指针的位置。

TPoint PosToPoint(int pos);

重载 TSlider 的成员函数，将指示指针的位置转换为坐标值。

A.10　TVSlider 类

头文件：slider.h

描述：作为 TSlider 类的派生类，TVSlider 提供了纵向 slider 控件的具体特性，它在 OWL 5.0 中的位置如图A-8所示。

Public 构造函数：

```
TVSlider(TWindow * parent,
         int id,
         int X, int Y, int W, int H,
         TResId thumbResId = IDB_VSLIDERTHUMB,
         TModule * module = 0);
```

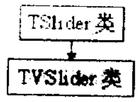


图 A-8　TVSlider 类在 OWL 5.0 中的位置

Protected 成员函数：

int HitTest(TPoint& point);

重载 TSlider 的成员函数，获得指定的坐标在 slider 控件上的对应值。

void NotifyParent(int scrollCode, int pos=0);

重载 TSlider 的成员函数，给父窗口发出 WS_VSCROLL 消息。

void PaintRuler(TDC&);

重载 TSlider 的成员函数，绘制内建标尺。

void PaintSlot(TDC&);

重载 TSlider 的成员函数，绘制轨道。

int PointToPos(TPoint& point);

重载 TSlider 的成员函数，将坐标值转换为指示指针的位置。

TPoint PosToPoint(int pos);

重载 TSlider 的成员函数，将指示指针的位置转换为坐标值。

A.11　TGauge 类

头文件：gauge.h

描述：作为 TControl 类的派生类，TGauge 类定义了一系列 gauge 控件的基本行为。gauge 对象通过横向或纵向的条状矩形显示一个过程的进展程度，在安装软件中常可见到这类控件。一般，我们用横向控件表示一个过程的状态，用纵向 gauge 控件表示信息分析的结果。TGauge 在 OWL 5.0 中的位置见图A-9。

Public 构造函数：

```
TGauge(TWindow * parent,
       const char far * title,
       int id,
       int X, int Y, int W, int H,
       bool isHorizontal = true,
       int margin = 0,
```

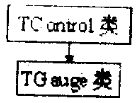


图 A-9　TGauge 类在 OWL 5.0 中的位置

```
TModule * module = 0);
```

Public 成员函数：

void GetRange(int& min, int& max) const;

返回 gauge 控件的范围。

int GetValue() const;

返回 gauge 控件当前的值。

void SetColor(TColor color);

设置数据成员 BarColor 为指定的颜色。

void SetLed(int spacing, int thick = 90);

设置数据成员 LedSpacing 和 LedThick 为指定的值。

void SetRange(int min, int max);

设置 gauge 控件的范围。

void SetValue(int value);

设置 gauge 控件当前的值。

Protected 数据成员：

TColor BarColor;

条状区域的颜色，缺省时为蓝色。

int IsHorizontal;

如果 gauge 控件是横向控件，值为 true，否则为 false。

int LedSpacing;

gauge 控件中条状图的间距。

int LedThick;

gauge 控件中条状图的厚度。

int Margin;

gauge 控件的边界宽度和高度。

int Max;

gauge 控件范围的最大值。

int Min;

gauge 控件范围的最小值。

int Value;

gauge 控件的当前值。

Protected 成员函数：

bool EvEraseBkgnd(HDC);

重载窗口类的同名函数，擦除背景。

void Paint(TDC& dc, bool erase, TRect& rect);

绘制整个 gauge 控件。

virtual void PaintBorder(TDC& dc);

绘制 gauge 控件的边框。

事件响应表：见表A-27。

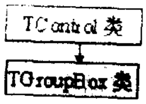
表A-27

EV_ macros	成员函数
EV_WM_ERASEBKGNDEvEraseBkgnd	

A.12 TGroupBox类

头文件：groupbox.h

描述：通常TGroupBox对象不是在对话框中使用，而是用来建立窗口中的一个组框。表面上看组框只是可视化地将一组选择(比如，检查框、radio按钮和其它控件)框起来。实际上，组框扮演着一个非常重要的角色，它处理组框中控件的状态变化(通常是检查框和radio按钮)。TGroupBox数在OWL 5.0中的位置如图A-10所示。



图A-10 TGroupBox类在OWL 5.0中的位置

Public 数据成员：

bool NotifyParent;

这个标志表明在组框中的选项改变时是否通知父窗口。缺省时，NotifyParent的值是true。

Public 构造函数：

```
TGroupBox(TWindow * Parent,
          int Id,
          const char far * text,
          int x, int y, int w, int h,
          TModule * module = 0);
TGroupBox(TWindow * Parent,
          int resourceId,
          TModule * module = 0);
```

Public 成员函数：

char far * GetClassName();

GetClassName函数返回TGroupBox窗口的注册类名"BUTTON"。如果启动了BWCC，GetClassName函数返回"BUTTON_CLASS"。

virtual void SelectionChanged(int controlId);

如果NotifyParent的值为true，SelectionChanged函数通过发出child-ID-based消息通知组框的父窗口组框中的一个选项改变了。这个成员函数可重新定义，以允许组框处理其中的控件选项改变事件。

A.13 TStatic 类

头文件：static.h

描述：TStatic 类是代表静态文本界面元素的界面类。静态元素包括了用户不能改变的文本或图形。当然，应用程序可以修改静态控件。如果你想要在父窗口对象中创建静态控件来显示诸如 copyright 之类的文字，你就需要用到 TStatic 对象。TStatic 类也可以用来改变对话框对象中的静态控件文本，它在 OWL 5.0 中的位置如图 A-11 所示。

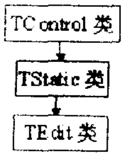


图 A-11 TStatic 类在 OWL 5.0 中的位置

Public 数据成员：

uint TextLen;

静态控件的文本缓冲区大小。因为在字符串的最后要加上一个 null，所以静态控件所能存储的字符串长度是 TextLen－1。TextLen 也表示静态控件对象所能传输的字节数。

Public 构造函数：

```
TStatic(TWindow * parent,
        int id,
        const char far * title,
        int x, int y, int w, int h,
        uint textLen = 0,
        TModule * module = 0);
TStatic(TWindow * parent,
        int resourceId,
        uint textLen = 0,
        TModule * module = 0);
```

Public 成员函数：

virtual void Clear();

清除与静态控件相关的字符串。

int GetText(char far * str, int maxChars);

获得静态控件的文本，将它存储在 maxChars 指定大小的 str 字符串中，并返回拷贝的字符数。

int GetTextLen();

返回静态控件的文本长度。

void SetText(const char far * str);

设置静态控件的文本为 str 指定的字符串。

uint Transfer(void * buffer, TTransferDirection direction);

从传输缓冲区 buffer 中传入或传出静态控件的文本。数据从缓冲区传入还是传出到缓冲区是由 direction 参数决定的。如果 direction 是 tdGetData，数据从缓冲区中传出；如果 direction 是 tdSetData，数据从缓冲区中传入；Transfer 总是返回传输数据的大小。如果只想得到这个结构的大小而不想传入或传出数据，就设置 direction 的值为 tdSizeData。

Protected 成员函数：

void EvSize(uint sizeType, TSize& size);

重载 TWindow 的同名函数。

virtual char far * GetClassName();

返回 TStatic 类的注册类名"STATIC"；或者，如果开启了 BWCC，就返回"STATIC_CLASS"。

事件响应表：见表 A－28

表 A－28

EV_ macros	成员函数
EV_WM_SIZE	EvSize

A.14　TEdit 类

头文件：edit.h

描述：TEdit 是代表编辑框控件界面元素的界面类。TEdit 对象在父窗口中创建一个编辑框控件。在对话框中创建 TEdit 对象有利于增加编辑框控件与应用程序的联系。TEdit 是可流类，它在 OWL 5.0 中的位置如图A－12所示。

编辑框控件有两种风格：单行和多行编辑框控件。多行编辑框控件允许编辑多行文本。编辑框控件中的第一个字符的位置为 0。对于多行编辑框控件，字符的位置是一行一行连续排列的。大多数的 TEdit 的成员函数管理着编辑框控件的文本，TEdit 还具有一些成员函数，它们自动响应编辑框控件的父窗口的菜单项选择，包括 cut、copy、和 paste。

Public 构造函数：

```
TEdit(TWindow * parent,
        int Id,
        const char far * text,
        int x, int y, int w, int h,
        uint textLen = 0,
        bool multiline = false,
        TModule * module = 0);
TEdit(TWindow * parent,
        int resourceID,
        uint textLen = 0,
        TModule * module = 0);
```

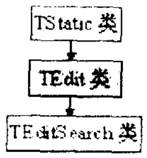


图 A－12　TEdit 类在 OWL 5.0 中的位置

Public 成员函数：

bool CanUndo();

如果可以取消最后一步编辑操作，返回 true。

void Clear();

重载 TStatic 类的成员函数，清除所有的文本。

void ClearModify();

　　重新设置编辑框控件的“改变标志”使得 IsModified 返回 false。

void Copy();

　　将选择的文本拷贝到剪贴板中。

void Cut();

　　删除选择的文本并拷贝到剪贴板中。

bool DeleteLine(int lineNumber);

　　删除编辑框控件中由 lineNumber 指定的一行文本，如果传入的参数为－1，则删除当前行。DeleteLine 删除整个一行且不影响其它行，删除成功后就返回 true，如果 lineNumber 为－1，但超出了范围或者出错，就返回 false。

bool DeleteSelection();

　　删除选择的文本，如果没有选择文本就返回 false。

bool DeleteSubText(uint startPos, uint endPos);

　　删除 startPos 和 endPos 参数设置的起止范围中的文本，如果成功就返回 true。

void EmptyUndoBuffer();

　　如果在编辑框控件中的一个操作可以被撤销，则编辑框控件就设置“undo”标志。EmptyUndoBuffer 函数重新设置或清除这个标志。

void FormatLines(bool addEOL);

　　设置 end－of－line 字符是否可以被增加或删除。

int GetFirstVisibleLine() const;

　　给出在编辑框控件中的最上面一行。对于单行编辑框控件，返回值为 0；对于多行编辑框控件则返回编辑框控件中的最上面一行的索引。

HLOCAL GetHandle() const;

　　返回控件所拥有的数据句柄。

bool GetLine(char far * str, int strSize, int lineNumber);

　　从编辑框控件中获得一行文本，并通过字符串 str 返回，如果不能获得文本或缓冲区不够，返回 false。strSize 表明返回了多少字符。

int GetLineFromPos(uint charPos);

　　从一个多行的编辑框控件中返回 charPos 指定位置的字符所在行的行数。

uint GetLineIndex(int lineNumber);

　　GetLineIndex 从多行编辑框控件中返回 lineNumber 指定行之前的字符数。

int GetLineLength(int lineNumber);

　　GetLineLength 从多行编辑框控件中返回 lineNumber 指定行中的字符数。

int GetNumLines();

　　返回一个多行编辑框控件的行数。

uint GetPasswordChar() const;

　　返回代替用户键入字符的显示字符。当编辑框控件的风格中包含 ES_PASSWORD，那么缺省的显示字符是(*)。

void GetRect(TRect& frmtRect) const;

　　获得一个多行编辑框控件的区域。

void GetSelection(uint& startPos, uint& endPos);

　　返回当前选择文本的开始位置 startPos 和结束位置 endPos。通过使用 GetSelection 函数和 GetSubText 函数，你可以获得当前的选择文本。

void GetSubText(char far * str, uint startPos, uint endPos);

获得一个编辑框控件中指定开始位置 startPos 和结束位置 endPos 之间的字符串。

EDITWORDBREAKPROC GetWordBreakProc() const;

获得当前的 wordwrap 功能。

void Insert(const char far * str);

将 str 指定的字符串插入到编辑框控件的当前插入点，代替所有当前选择的文本。

bool IsModified();

如果用户改变了编辑框控件中的文本，就返回 true。

bool IsValid(bool reportErr = false);

当编辑框控件中的文本有效时返回 true。

char far * LockBuffer(uint newsize=0);

锁定编辑框控件的缓冲区并返回一个缓冲区的指针。输入参数 newsize 大于 0 将使缓冲区重新按照 newsize 来定义大小。LockBuffer 函数之后，你必须调用 Unlock 函数。

void Paste();

将剪贴板中的文本插入到编辑框控件的当前插入点。

void Scroll(int horizontalUnit, int verticalUnit);

根据 horizontalUnit 和 verticalUnit 定义的字符数滚动一个多行编辑框控件的滚动条。

```
int Search(uint startPos,
           const char far * text,
           bool caseSensitive=false,
           bool wholeWord=false,
           bool up=false);
```

执行一个大小写敏感或不敏感的文本查询。

void SetHandle(HLOCAL localMem);

设置多行编辑框控件保存内容的文本缓冲区的句柄。

void SetPasswordChar(uint ch);

设置代替用户键入字符的显示字符。当编辑框控件的风格中包含 ES_PASSWORD，那么缺省的显示字符是(*)。

void SetReadOnly(bool readOnly);

设置编辑框控件是只读的或是可写的。

void SetRect(const TRect& frmtRect);

设置多行编辑框控件的区域。

void SetRectNP(const TRect& frmtRect);

设置多行编辑框控件的区域，但不重画控件。

bool SetSelection(uint startPos, uint endPos);

设置选择的文本，起始点为 startPos 结束点为 endPos，但不包括 endPos 所在的字符。

void SetTabStops(int numTabs, const int far * tabs);

设置 Tab 键的位置。

void SetValidator(TValidator * validator);

在编辑框控件中建立有效对象。

void SetWordBreakProc(EDITWORDBREAKPROC proc);

在多行编辑框控件中，SetWordBreakProc 表明一个应用程序提供的 word－break 功能代替了缺省的 word－break 功能。

uint Transfer(void * buffer TTransferDirection direction);

传输 TEdit 的信息。

void Undo();

撤销最后一步操作。

void UnlockBuffer(const char far * buffer, bool updateHandle=false);

开启编辑框控件的缓冲区。如果内容改变了，updateHandle 就设置为 true。

void ValidatorError();

保存错误信息。

Protected 数据成员：

TValidator * Validator;

指向一个有效对象。

Protected 成员函数：

bool CanClose();

在关闭窗口之前检查是否所有的窗口都可以关闭。如果任何一个子窗口返回 false，CanClose 函数就返回 false 并终止关闭过程。如果所有的子窗口都返回 true，则 CanClose 返回 true。

void CmCharsEnable(TCommandEnabler& commandHandler);

决定菜单项 Clear 的功能是否对当前选择的文本开启。

void CmEditClear();

通过调用 Clear 函数自动响应一个菜单项的选择 CM_EDITCLEAR。

void CmEditCopy();

通过调用 Copy 函数自动响应一个菜单项的选择 CM_EDITCOPY。

void CmEditCut();

通过调用 Cut 函数自动响应一个菜单项的选择 CM_EDITCUT。

void CmEditDelete();

通过调用 DeleteSelection 函数自动响应一个菜单项的选择 CM_EDITDELETE。

void CmEditPaste();

通过调用 Paste 函数自动响应一个菜单项的选择 CM_EDITPASTE。

void CmEditUndo();

通过调用 Undo 函数自动响应一个菜单项的选择 CM_EDITUNDO。

void CmModEnable(TCommandEnabler& commandHandler);

决定菜单项 Undo 的功能是否对当前选择的文本开启，

void CmPasteEnable(TCommandEnabler& commandHandler);

决定菜单项 Paste 的功能是否对当前选择的文本开启。

void CmSelectEnable(TCommandEnabler& commandHandler);

决定菜单项 Cut、Copy 或 Delete 的功能是否对当前选择的文本开启。

void ENErrSpace();

响应一个出错信息，发出警告的声响。

void EvChar(uint key, uint repeatCount, uint flags);

确认输入的文本。

uint EvGetDlgCode(MSG far * msg);

响应 WM_GETDLGCODE 消息。

void EvKeyDown(uint key, uint repeatCount, uint flags);

EvKeyDown 将一个虚键解释成一个动作。

void EvKillFocus(HWND hWndGetFocus);

响应一个 WM_KILLFOCUS 消息，使窗口失去焦点。

char far * GetClassName();

返回 TEdit 的注册类名"EDIT"。

void SetupWindow();

如果数据成员 textLen 不是 0，SetupWindow 限制编辑框控件可以输入的字符数为 textLen －1。

事件响应表：见表 A-29

表 A-29

EV_ macros	成员函数
EV_COMMAND (CM_EDITCLEAR, CmEditClear)	CmEditClear
EV_COMMAND (CM_EDITCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND (CM_EDITCUT, CmEditCut)	CmEditCut
EV_COMMAND (CM_EDITDELETE, CmEditDelete)	CmEditClear
EV_COMMAND (CM_EDITPASTE, CmEditPaste)	CmEditPaste
EV_COMMAND (CM_EDITUNDO, CmEditUndo)	CmEditUndo
EV_COMMAND_ENABLE(CM_EDITCLEAR, CmCharsEnable)	CmCharsEnable
EV_COMMAND_ENABLE(CM_EDITCOPY, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITCUT, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITDELETE, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITPASTE, CmPasteEnable)	CmPasteEnable
EV_COMMAND_ENABLE(CM_EDITUNDO, CmModEnable)	CmModEnable
EV_NOTIFY_AT_CHILD (EN_ERRSPACE, ENErrSpace)	ENErrSpace
EV_WM_CHAR	EvChar
EV_WM_GETDLGCODE	EvGetdlgcode
EV_WM_KEYDOWN	EvKeydown
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_CHILDINVALID	EvChildInvalid

A.15 TEditSearch 类

头文件：editsear.h

描述：TEditSearch 是一个响应菜单项命令 Find、Replace 和 FindNext 的编辑框控件，它是可流类，在 OWL 5.0 中的位置如图 A-13 所示。

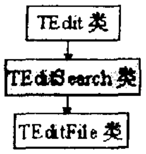


图 A-13　TEditSearch 类在 OWL 中的位置

Public 数据成员：

uint SearchCmd;

search 命令打开的对话框的 ID 值。

TFindReplaceDialog::TData SearchData;

SearchData 结构定义了查询的文本字符串，代替的字符串和文本缓冲区的大小。

TFindReplaceDialog * SearchDialog;

查找和替换对话框的信息(比如，要查找和替换的文本)和检查框的设置。

Public 构造函数：

```
TEditSearch(TWindow * parent = 0,
            int Id = 0,
            const char far * text = 0,
            int x = 0, int y = 0, int w = 0, int h = 0,
            TModule * module = 0);
```

Public 成员函数：

void CmEditFind();

响应查找命令 CM_EDITFIND，打开一个 TFindDialog 对话框。

void CmEditFindNext();

响应 FindNext 命令 CM_EDITFINDNEXT，调用 DoSearch 函数重复查找操作。

void CmEditReplace();

响应替换命令 CM_EDITREPLACE，打开一个 TReplaceDialog 对话框。

void DoSearch();

根据 SearchData 中的信息执行一个操作或替换操作。

LRESULT EvFindMsg(WPARAM, LPARAM);

响应查找替换无模式对话框发出的消息，调用 DoSearch 函数继续查找。

void SetupWindow();

发出 CM_EDITFIND 或 CM_EDITREPLACE 消息来重新打开一个查找替换无模式对话框。调用 TEdit::SetupWindow。

事件响应表：见表 A-30

表 A-30

EV_ macros	成员函数
EV_COMMAND(CM_EDITFIND, CmEditFind)	CmEditFind
EV_COMMAND(CM_EDITFINDNEXT, CmEditFindNext)	CmEditFindNext
EV_COMMAND(CM_EDITREPLACE, CmEditReplace)	CmEditReplace
EV_REGISTERED(FINDMSGSTRING, EvFindMsg)	EvFindMsg

A.16　TEditFile 类

头文件：editfile.h

描述：TEditFile 是一个文件编辑窗口。TEditFile 的数据成员和成员函数管理着文件对话框，并自动响应文件命令，比如，Open、Read、Write、Save 和 SaveAs。TEditFile 是可流类，它在 OWL 5.0 中的位置如图 A-14 所示。

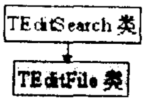


图 A-14　TEditFile 类在 OWL 5.0 中的位置

Public 数据成员：

TOpenSaveDialog::TData FileData;

用户打开和保存文件的信息。

char far * FileName;

正再编辑的文件的名字。

Public 构造函数和析构函数：

```
TEditFile(TWindow * ()t = 0,
          int Id = 0,
          const char far * text = 0,
          const char far * fileName = 0,
          TModule * module = 0);
~TEditFile();
```

Public 成员函数：

virtual bool CanClear();

如果与文本相连的编辑框控件可以清除则返回 true。

virtual bool CanClose();

如果文件编辑窗口可以关闭则返回 true。

void CmFileNew();

调用 NewFile 函数来响应一个命令 CM_FILENEW。

void CmFileOpen();

调用 Open 函数来响应一个命令 CM_FILEOPEN。

void CmFileSave();

调用 Save 函数来响应一个命令 CM_FILESAVE。

void CmFileSaveAs();

调用 SaveAs 函数来响应一个命令 CM_FILESAVEAS。

void NewFile();

开始编辑新的文件。先调用 CanClear 函数决定正在编辑的文件是否可以安全清除。

Open();

打开一个文件。先调用 CanClear 函数决定正在编辑的文件是否可以安全清除，如果返回是 true，打开一个文件对话框接收一个用户键入的文件名。用新的文件名作为传入参数调用 ReplaceWith 函数。

bool Read(const char far * fileName=0);

将 fileName 指定的文件读入到编辑窗口中。如果读入成功则返回 true。

void ReplaceWith(const char far * fileName);

调用 SetFileName 函数和 Read 函数读入 fileName 指定的文件来代替当前编辑的文件。

bool Save();

将编辑窗口中的内容存入文件，如果 Editor—>IsModified 返回 false，Save 返回 true，表明在最后一次打开文件或保存文件以后没有任何改动。

bool SaveAs();

将编辑窗口中的内容存入用户所给的文件中，打开一个文件保存对话框，接收用户输入的文件名。如果用户按下了 OK 键，SaveAs 就调用 SetFileName 函数和 Write 函数。如果文件成功存盘就返回true。

void SetFileName (const char far * fileName);

设置 FileName 并更新窗口的标题条。

bool Write(const char far * fileName=0);

将编辑窗口中的内容保存到 FileName 指定的文件中。如果写文件成功就返回 true。

Protected 成员函数：

void SetupWindow();

创建一个编辑窗口。调用 TEditFile::SetupWindow，如果 FileName 存在就设置窗口的标题条为 FileName，否则就设置为"Untitled"。

事件响应表：见表 A - 31

表 A - 31

EV_ macros	成员函数
EV_COMMAND (CM_FILENEW, CmFileNew)	CmFileNew
EV_COMMAND (CM_FILEOPEN, CmFileOpen)	CmFileOpen
EV_COMMAND (CM_FILESAVE, CmFileSave)	CmFileSave
EV_COMMAND (CM_FILESAVEAS, CmFileSaveAs)	CmFileSaveAs

A.17　TButton 类

头文件：button.h

描述：TButton 是代表按钮界面元素的界面类。在 TWindow 对象窗口中，你必须使用 TButton 类在父窗口中创建一个按钮控件，你也可以在对话框中采用 TButton 对象用来建立应用程序与按钮之间的联系。TButton 类是可流类，它在 OWL 5.0 中的位置如图 A - 15 所示。

按钮有两种类型：规则的和缺省的。规则按钮具有细的边框，而缺省按钮具有粗的边框，它代表窗口的缺省动作。

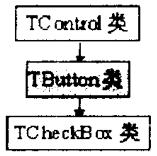


图 A - 15　Button 类在 OWL 5.0 中的位置

Public 数据成员：

bool IsDefPB;

指明按钮是否被认为是缺省按钮。

Public 构造函数：

```
TButton(Window * parent,
        int Id,
        const char far * text,    int X, int Y, int W, int H,
        bool isDefault = false,
        TModule * module = 0);
TButton(TWindow * parent,
        int resourceID,
        TModule * module = 0);
```

Protected 数据成员：

bool IsCurrentDefPB;

指明当前的按钮是否被认为是缺省按钮。

Protected 成员函数：

LRESULT BMSetStyle(WPARAM, LPARAM);

因为一个按钮不能同时是自绘按钮又具有 push 按钮风格，因此按钮是自绘按钮而系统试图将它设置为 BS_DEFPUSHBUTTON 风格时，BMSetStyle 函数就跟踪按钮的风格。如果按钮的风格是 BS_DEFPUSHBUTTON，BMSetStyle 函数设置 CurrentDefPB 的值为 true。

uint EvGetDlgCode(MSG far *);

响应发给与控件相关对话框的 WM_GETDLGCODE 消息。

char far * GetClassName();

重载 TWindow 的 GetClassName 函数。如果 BWCC 开启，就返回 TButton 的注册类名"BUTTON_CLASS"；如果 BWCC 没有开启，就返回"BUTTON"。

void SetupWindow();

重载 TWindow 的 SetupWindow 函数。如果按钮是缺省的 push 按钮或自绘按钮，SetupWindow 就发出一个 DM_SETDEFID 消息到主窗口。

事件响应表：见表 A-32

表 A-32

EV_ macros	成员函数
EV_WM_GETDLGCODE	EVGetDlgCode
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle

A.18　TCheckBox 类

头文件：checkbox.h

描述：TCheckBox 类是代表检查框控件的类，它是可流类，在 OWL 5.0 中的位置如图 A-16 所示。在主窗口中可以用 TCheckBox 来创建一个检查框控件，也可以在对话框中使用 TCheckBox 对象。

双态检查框的状态有选择状态和非选择状态；三态检查框还增加了一种灰色状态。TCheckBox 类的成员函数使你可以轻松的控制检查框的状态。检查框也可以被包含在一个组框中。

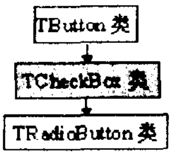


图 A-16　TCheckBox 类在 OWL 5.0 中的位置

Public 数据成员：

TGroupBox ＊Group；

如果检查框属于一个组框(一个 TGroupBox 对象)，Group 就指向该组框对象。如果检查框不在组框中，Group 的值就是 0。

Public 构造函数：

```
TCheckBox(TWindow * parent,
        int Id,
        const char far * title,
        int x, int y, int w, int h,
        TGroupBox * group = 0,
        TModule * module = 0);
TCheckBox(TWindow * parent,
        int resourceId,
        TGroupBox * group = 0,
        TModule * module = 0);
```

Public 成员函数：

void Check()；

通过调用 SetCheck 函数传输 BF_CHECKED 值来设置检查框的状态为选择状态。如果检查框在一个组框中，就通知组框；检查框的状态改变了。

uint GetCheck() const；

返回检查框的状态，检查框的状态返回值如下：

Checked	BF_CHECKED
Unchecked	BF_UNCHECKED
Grayed	BF_GRAYED

uint GetState() const；

返回检查框的选择、焦点和高亮状态。

void SetCheck(uint check)；

按照 check 的值设置检查框的状态。

void SetState(uint state)；

设置检查框的选择、焦点和高亮状态。

voidSetStyle(uint style, bool redraw)；

改变检查框的风格。

void Toggle()；

改变检查框的状态：如果是双态检查框，就在选择和非选择状态之间更替；如果是三态检查框，就在选择、非选择和灰色三种状态之间更替。

uint Transfer(void * buffer, TTransferDirection direction);

重载 TWindow::Transfer 函数，传输检查框的选择状态到缓冲区中或反之。如果 direction 的值是 tdSetData，设置检查框的状态为传输缓冲区中的值；如果 direction 的值是 tdGetData，把当前的检查框状态传输到缓冲区。Transfer 总是返回传输数据的大小。如果只想得到这个结构的大小而不想传入或传出数据，就设置 direction 的值为 tdSizeData。

Protected 成员函数：

void BNClicked();

响应广播事件 BN_CLICKED，表明用户点了检查框。如果 Group 的值不为 0，BNClicked 函数就调用组框的成员函数 SelectionChanged 来通知组框：检查框的状态改变了。

uint EvGetDlgCode(MSG far * msg);

重载 TButton 类的成员函数来响应 WM_GETDLGCODE 消息。

char far * GetClassName();

如果 BWCC 开启，返回"CHECK_CLASS"，如果没有开启 BWCC，则返回"BUTTON"。

事件响应表：见表 A-33。

表 A-33

EV_ macros	成员函数
EV_NOTIFY_AT_CHILD (BN_CLICKED, BNClicked)	BNClicked
EV_WM_GETDLGCODE	EVGetDlgCode

A.19　TRadioButton 类

头文件：radiobut.h

描述：TRadioButton 类代表窗口中相应的 radio 按钮控件的界面元素，它在 OWL 5.0 中的位置如图 A-17所示。在父窗口中可以使用 TRadioButton 类来创建 radio 按钮控件，也可以在对话框中使用 TRadioButton 对象，使得 radio 按钮控件与应用程序更好的联系。

radio 按钮有两种状态：选择和非选择状态。TRadioButton 类继承了它的基类 TCheckBox 的状态管理成员函数。另外，radio 按钮可以是一个组框的项目，组框可视化地和逻辑地将其中的控件归为一组。

Public 构造函数：

```
TRadioButton(TWindow * parent,
             int id,
             const char far * title,
             int x, int y, int w, int h,
             TGroupBox * group = 0,
             TModule * module = 0);
TRadioButton(TWindow * parent,
             int resourceId,
```

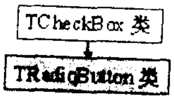


图 A-17　TRadroButton 类在 OWL 5.0 中的位置

```
        TGroupBox * group,
        TModule * module = 0);
```

Protected 成员函数：

void BNClicked();

相应一个输入的 BN_CLICKED 消息。

char far * GetClassName();

返回 radio 按钮类的注册类名“BUTTON”。

事件响应表： 见表 A-34

表 A-34

EV_ macros	成员函数
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle
EV_WM_GETDLGCODE	EvGetDlgCode

A.20 TVbxControl 类

头文件： vbxctl.h

描述： TVbxControl 类从 TControl 中派生出来，它提供了 Visual Basic 的控件 VBX 的接口。可用于设置或获得 VBX 的参数。16 位和 32 位的应用程序都可以用 VBX 控件。TVbxControl 类在 OWL 5.0 中的位置见图A-18。

你可以使用索引值或名字来操作 VBX 控件的参数，为了访问不同类型的参数，TVbxControl 类提供了一系列重载的 GetProp 函数；为了设置不同类型的参数，提供了一系列重载的 SetProp 函数。

TVbxControl 类提供了事件处理函数，执行比较、删除、绘制和测量等 VBX 控件的请求。如果你想要 TVbxControl 对象处理增加的 VBX 事件，可以从 TVbxControl 类中派生出一个新的类，在事件响应表中增加你想处理的事件。

Public 构造函数和析构函数：

```
TVbxControl(TWindow * parent,
        int id,
        const char far * vbxName,
        const char far * vbxClass,
        const char far * title,
        int x, int y, int w, int h,
        long initLen = 0,
        void far * initData = 0,
        TModule * module = 0);
TVbxControl(TWindow * parent,
        int resourceId,
```

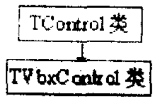


图 A-18　TVbxControl 类在 OWL 5.0 中的位置

```
TModule * module = 0);
~TVbxControl();
```

Public 成员函数：

bool AddItem(int index, const char far * item);

按照 index 所指的索引在 VBX 控件列表中增加一项 item。如果成功就返回 0。

bool Drag(int action);

根据 action 的值控制 VBX 的拖放状态。action 的值为 0 时，放弃一个拖动操作；action 的值为 1 时，开始拖动一个控件；action 的值为 2 时，结束拖动一个控件；

int GetEventIndex(const char far * name);

返回与事件名 name 相关的索引，如果不成功就返回 -1。

const char far * GetEventName(int eventindex);

返回与事件索引相关的事件名字符串，如果出错就返回 0。

HCTL GetHCTL();

返回与 TVbxControl 对象相对应的 VBX 控件的句柄。

int GetNumEvents();

返回与 VBX 控件相关的事件数目。

int GetNumProps();

返回与 VBX 控件相关的参数数目。

```
bool GetProp(int propIndex, int& value, int arrayIndex = -1);
bool GetProp(int propIndex long& value, int arrayIndex = -1);
bool GetProp(int propIndex, ENUM& value, int arrayIndex=-1);
bool GetProp(int propIndex, HPIC& value, int arrayIndex=-1);
bool GetProp(int propIndex float& value, int arrayIndex = -1);
bool GetProp(int propIndex string& value, int arrayIndex = -1);
bool GetProp(int propIndex, COLORREF value, int arrayIndex = -1);
bool GetProp(const char far * name, int& value, int arrayIndex = -1);
bool GetProp(const char far * name, long& value, int arrayIndex = -1);
bool GetProp(const char far * name, float& value, int arrayIndex = -1);
bool Getprop(const char far * name, ENUM& value, int arrayIndex=-1);
bool Getprop(const char far * name, HPIC& value, int arrayIndex=-1);
bool GetProp(const char far * name, string& value, int arrayIndex = -1);
bool GetProp(const char far * name, COLORREF value, int arrayIndex = -1);
```

以上是获得参数的一系列函数。

int GetPropIndex(const char far * name);

获得 name 指定的参数名的索引，如果出错就返回 -1。

const char far * GetPropName(int propIndex);

获得 index 指定参数索引的参数名，如果出错就返回 0。

int GetPropType(int propIndex);

通过参数索引获得参数的类型，如果出错就返回 0。表 A-35 列出了参数和对应的 C++ 类型。

表 A-35

参数类型	C++类型
PTYPE_CSTRING	HSZ
PTYPE_SHORT	short
PTYPE_LONG	int32
PTYPE_BOOL	bool
PTYPE_COLOR	uint32 or COLORREF
PTYPE_ENUM	uint8 or ENUM
PTYPE_REAL	float
PTYPE_XPOS	int32 (Twips)
PTYPE_XSIZE	int32(Twips)
PTYPE_YPOS	int32 (Twips)
PTYPE_YSIZE	int32 (Twips)
PTYPE_PICTURE	HPIC
PTYPE_BSTRING	HLSTR

int GetPropType(char far * name);

　　通过参数名获得参数的类型。如果出错就返回 0。

bool IsArrayProp(int propIndex);

　　如果 propIndex 指定的参数是一个数组就返回 true。

bool IsArrayProp(char far * name);

　　如果 name 指定的参数是一个数组就返回 true。

bool Method(int method, long far * args);

　　此函数用来调用用户方式。如果一个 VBX 控件可以以 method 方式响应就返回 true。

bool Move(long x, long y, long w, long h);

　　按照所给的屏幕坐标(x,y)移动 VBX 控件(坐标的原点在屏幕左上角)，也可按照所给的宽度 w 和高度 h 改变 VBX 控件的对象。如果成功就返回非 0 值。

bool Refresh();

　　重画控件区域。

bool RemoveItem(int index);

　　删除 index 指定的项目。比如可以从一个列表框、组合框或数据库中删除项目。

bool SetProp(int propIndex, int value, int arrayIndex = -1);

bool SetProp(int propIndex, long value, int arrayIndex = -1);

bool SetProp(int propIndex, ENUM value, int arrayIndex=-1);

bool SetProp(int propIndex, HPIC value, int arrayIndex=-1);

bool SetProp(int propIndex, float value, int arrayIndex = -1);

bool SetProp(int propIndex, const string& value, int arrayIndex = -1);

bool SetProp(int propIndex, const char far * value, int arrayIndex = -1);

bool SetProp(int propIndex, COLORREF value, int arrayIndex = -1);

bool SetProp(const char far * name, int value, int arrayIndex = -1);

bool SetProp(const char far * name, long value, int arrayIndex = -1);

bool SetProp(const char far * name, ENUM value, int arrayIndex=-1);

bool SetProp(const char far * name, HPIC value, int arrayIndex = -1);

bool SetProp(const char far * name, float value, int arrayIndex = -1);

bool SetProp(const char far * name, const string& value, int arrayIndex = -1);

bool SetProp(const char far * name, const char far * value, int arrayIndex = -1);

bool SetProp(const char far * name, COLORREF value, int arrayIndex = -1);

以上为一系列参数设置函数。

void SetupWindow();

一个 VBX 控件有一个 HWND 加一个 VBX 句柄。通常，VBX 控件句柄首先创建，但如果窗口已经创建，你可以使用这个函数来增加 VBX 控件句柄。

Protected 成员函数：

char far * GetClassName();

获得一个 VBX 的注册类名。

bool GetVBXProperty(int propIndex, void far * value, int arrayIndex = -1);

如果指定的参数存在就返回非 0 值，并获得此参数。

void PerformCreate(int menuOrId);

创建一个与 VBX 控件相联系的新的控件窗口。创建控件的 ID 值、VBX 控件名和类以及窗口的标题。

bool SetVBXProperty(int propIndex, int32 value, int arrayIndex=-1);

如果指定的参数存在就返回非 0 值，并设置此参数。

A.21　TListBox 类

头文件： listbox.h

描述： 这个类代表一个与之相关的列表框控件界面元素的界面对象，在窗口对象中创建一个列表框控件的时候就需要使用 TListBox 类。在对话框中使用这个类可以使列表框控件与应用程序方便地交流。TListBox 类是可流类，它在 OWL 5.0 中的位置如图 A-19 所示。

Public 构造函数：

TListBox(TWindow * Parent,
　　int Id,
　　int x, *int* y, *int* w, *int* h,
　　TModule * module = 0);

TListBox(TWindow * Parent,
　　int resourceId,
　　TModule * module = 0)

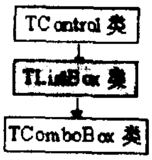


图 A-19　TListBox 类在 OWL 5.0 中的位置

Public 成员函数：

virtual int AddString(const char far * str);

在列表框中增加字符串，返回这个字符串在列表框中的位置。列表框中的第一个位置为 0，如果返回一个负数就说明出错。如果列表框控件的风格中包含 LBS_SORT，那么其中的项目就会自动排序。

virtual void ClearList();

清除列表框中的所有项目。

virtual int DeleteString(int index);

删除 index 指定的项目。如果删除成功就返回列表框中的剩下的项目数;如果出错就返回一个负数。

virtual int DirectoryList(uint attrs, const char far * fileSpec)

在列表框中增加一系列文件名。

int FindExactString(const char far * str, int searchIndex) const;

在列表框中查找与字符串 str 完全相同的项目。查找是从 searchIndex 指定的项目开始，如果匹配成功，就返回那个项目的索引值；如果没有找到项目，就返回一个负数。如果想要从列表框的第一个项目开始寻找，就将 searchIndex 的值设置为-1。

virtual int FindString(const char far * str, int Index) const;

与 FindExactString 功能相似，但 FindString 寻找的是第一个以字符串 str 开头的项目。

int GetCaretIndex() const;

返回当前焦点所在的列表框项目的索引。在单选框中，返回值是所选择的项目索引。

virtual int GetCount() const;

返回列表框中的项目数，如果出错就返回一个负数。

int GetHorizontalExtent() const;

获得列表框可以进行的横向滚动的像素点数目。

virtual uint32 GetItemData(int index) const;

返回 SetItemData 设置的一个列表框项目的 32 位值。

virtual int GetItemHeight(int index) const;

返回指定列表框项目的高度的像素点数目。

int GetItemRect(int index, TRect& rect) const;

返回指定的列表框项目在屏幕上的显示区域。

bool GetSel(int index) const;

返回指定的一个列表框项目的选择状态。如果返回值是 true，说明项目被选择了；否则说明项目没有被选择。

int GetSelCount() const;

返回单选框或多选框中所选择的项目数。

virtual int GetSelIndex() const;

用于单选框，如果有项目被选择就返回项目的非负索引，如果选择项目就返回一个负数。

int GetSelIndexes(int * indexes, int maxCount) const;

用于多选框，如果有项目被选择就在 indexes 数组中填充所选择的项目索引并返回 indexes 数组中的元素个数。maxCount 决定了选择的最大数目。

int GetSelString(char far * str, int maxChars) const;

获得当前的选择项目，将它拷贝到字符串 str 中。对于单选框，通常 GetSelString 返回字符串的长度，但如果出错就返回一个负数，如果没有项目选择就返回 1；如果是多选框就返回-1。maxChars 决定了所能接收的字符串的最大长度。

int GetSelStrings(char far * * strs, int maxCount, int maxChars) const;

获得多选框中被选择项目的总数，将项目的字符串拷贝到缓冲区中。str 指向一系列字符串，其中的每一个字符串的大小都是 maxChars 指定的，maxCount 指定 str 中可以拥有的字符串的最大数目。

virtual int GetString(char far * str, int index) const;

获得 index 所指定项目的字符串并将它拷贝到 str 中，若成功返回字符串的长度，否则返回一个负数。

virtual int GetStringLen(int Index) const;

返回 index 指定的项目的字符串长度，如果出错就返回一个负数。

int GetTopIndex() const;

返回显示在列表框最上面的项目索引。

virtual int InsertString(const char far * str, int index);

在列表框的指定位置插入一个字符串，返回这个项目在列表框中的实际位置，如果出错就返回一个负数。如果 index 的值为－1，就将项目插入到列表框的最后。

int SetCaretIndex(int index, bool partScrollOk);

将焦点设置到 index 指定的列表框项目。

void SetColumnWidth(int width);

设置列表框项目的宽度。

void SetHorizontalExtent(int horzExtent);

设置列表框可以进行的横向滚动的像素点数目。

virtual int SetItemData(int index, uint32 itemData);

设置 index 指定的列表框项目一个 32 的值。

virtual int SetItemHeight(int index, int height);

设置列表框项目的高度。

int SetItemRect(int index, TRect& rect) const;

设置列表框中当前显示的一个项目的区域。

int SetSel(int index, bool select);

对于多选框，设置 index 所指定项目的选择状态。

virtual int SetSelIndex(int index);

对于单选框，设置被选择的项目为 index 所指定的项目。如果 index 为－1，列表框将清除所有的选择。如果出错就返回一个负数。

int SetSelIndexes(int * indexes, int numSelections, bool shouldSet);

对于多选框，设置 indexes 数组中的项目的选择状态并返回函数所设置项目的数目，如果设置出错就返回一个负数。如果 shouldSet 的值为 true，设置项目为选择状态；如果 shouldSet 的值为 false，设置项目为非选择状态。如果 numSelections 的值为负数，就设置列表框中所有项目的选择状态。

int SetSelItemRange(bool select, int first, int last);

从 first 到 last 选择项目。

int SetSelString(const char far * str, int searchIndex);

对于单选框，从 SearchIndex 所指定的项目开始查找以 str 开头的项目，并选择它。如果 SearchIndex 的值为－1，就查找全部的项目。SetSelString 返回新选择的项目位置，如果出错就返回一个负数。

int SetSelStrings(const char far * * prefixes, int numSelections, bool shouldSet);

对于多选框，选择以 prefixes 字符串数组中所指定的字符串开头的项目。如果 shouldSet 的值为 true，设置匹配项目的状态为选择状态；如果 shouldSet 的值为 false，设置匹配项目的状态为非选择状态。如果设置出错就返回一个负数。

bool SetTabStops(int numTabs, int far * tabs);

设置 tab 键的站点。

int SetTopIndex(int index);

设置显示在列表框作上面的项目的索引。

uint Transfer(void * buffer, TTransferDirection direction);

传输列表框的项目和选择状态。buffer 指向一个 TListBoxData 结构。如果 direction 是 tdGetData，数据从

缓冲区中传出；如果 direction 是 tdSetData，数据从缓冲区中传入；Transfer 总是返回传输数据的大小。如果只想得到这个结构的大小而不想传入或传出数据，就设置 direction 的值为 tdSizeData。

Protected 成员函数：

char far * GetClassName()；

返回 TListBox 类的注册类名"LISTBOX"。

A.22 TListBoxData 结构

头文件：listbox. h

描述：用来传输列表框控件的内容。

Public 数据成员：

TDwordArray * ItemDatas；

列表框项目的一个 32 位值。

int SelCount；

被选择项目的数目。

TIntArray * SelIndices；

多选框中被选择项目的索引。

TStringArray * SelStrings；

多选框中被选择项目的字符串。

TStringArray * Strings；

指向一个传输给 TListBox 对象的字符串数组。

Public 构造函数和析构函数：

TListBoxData()；

~TListBoxData()；

Public 成员函数：

void AddString(const char * str, bool isSelected = false)；

在字符串数组 Strings 中增加一个指定的字符串。如果 IsSelected 的值为 true，就将字符串增加到 SelStrings 中，并增加 SelCount 的值。

void AddStringItem(const char * str, uint32 itemData, bool isSelected = false))；

在字符串数组 Strings 中增加一个指定的字符串，然后选择这个项目并就项目数据增加到数组 ItemDatas 中。

void Clear()；

重新设置列表框，将数组 ItemDatas 和 Strings 置空并调用 ResetSelections 函数。

TDwordArray& GetItemDatas()；

返回指向数组 ItemDatas 的指针。

TIntArray& GetSelIndices()；

返回指向数组 SelIndices 的指针。

void GetSelString(char far * buffer, int bufferSize, int index = 0) const;

根据 index 所给的位置在数组 SelStrings 中定位字符串并将它拷贝到 buffer 中。

int GetSelStringLength(int index = 0) const;

按照 index 指定的位置返回数组 SelStrings 中的一个字符串的长度。

TStringArray& GetStrings();

返回一个指向字符串数组的指针。

void ResetSelections();

从数组 SelStrings 中删除所有的字符串并设置 SelCount 的值为 0。

void Select(int index);

选择 index 所指定的字符串。

void SelectString(const char far * str);

将字符串 str 增加到数组 SelStrings 中并增加 SelCount 的值。

A.23　TComboBox 类

头文件：combobox.h

描述：TComboBox 类是 TListBox 类的派生类，继承了一系列父类的行为。TComboBox 类是可流类，在 OWL 5.0 中的位置如图 A-20 所示。

如果要在窗口对象中增加一个组合框就需要使用 TComboBox 类，与其他控件对象相同，在对话框对象中使用 TComboBox 对象可以使组合框与应用程序的联系更加简单。

有三种组合框可以选择，它们是由不同的风格参数决定的(见表 A-36)。

表 A-36

组合框类型	风格参数
简单组合框	CBS_SIMPLE
下拉编辑组合框	CBS_DROPDOWN
下拉静态组合框	CBS_DROPDOWNLIST

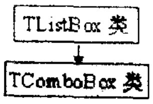


图 A-20　TComboBox 类在 OWL 5.0 中的位置

Public 数据成员：

uint TextLen;

与组合框相关的编辑框中文本的长度

Public 构造函数：

```
TComboBox(TWindow * parent,
          int id,
          int x, int y, int w, int h,
          uint32 style,
          uint textLen,
          TModule * module = 0);
TComboBox(TWindow * parent,
          int ResourceId,
          uint textLen = 0,
          TModule * module = 0);
```

Public 成员函数：

virtual int AddString(const char far * string);

在组合框的列表中增加一个字符串，返回这个字符串在列表中的索引，如果出错就返回一个负数。

void Clear();

清除编辑框中的内容。

virtual void ClearList();

清除列表中的所有项目。

virtual int DeleteString(int index);

从列表中删除 index 指定的字符串，返回删除后列表中剩下的项目数目，如果出错就返回一个负数。

virtual int DirectoryList(uint attrs, const char far * fileSpec);

把一个目录中的文件名增加到列表中。

virtual int FindString(const char far * find, int indexStart) const;

从 indexStart 指定的项目开始查找一个字符串。如果找到就返回这个项目的索引，如果出错就返回一个负数。

virtual int GetCount() const;

返回列表中的项目数目，如果出错就返回一个负数。

void GetDroppedControlRect(TRect& Rect) const;

返回下列列表框在屏幕的位置坐标。

bool GetDroppedState() const;

对于下列式组合框，返回列表框是否可见。

int GetEditSel(int& startPos, int& endPos);

返回编辑框中所选文本的开始和结束位置，如果组合框中没有编辑框就返回 CB_ERR。

bool GetExtendedUI() const;

决定组合框是否具有扩展用户界面，它与缺省的用户界面的不同之处在于：

* 如果用户点击静态文本区域就显示列表框；
* 如果用户按下 Down 按键就显示列表框；
* 如果列表是不可见的，就在静态文本区域中滚动。

如果组合框具有扩展用户界面就返回 true，否则返回 false。

virtual uint32 GetItemData(int index) const;

返回 index 所指定的项目的 32 值。

int GetItemHeight(int index) const;

返回列表中 index 指定项目的高度，如果出错就返回一个负数。

virtual int GetSelIndex() const;

返回所选择项目的索引，如果出错就返回一个负数。

virtual int GetString(char far * str, int index) const;

获得 index 指定的项目字符串，返回字符串的长度，如果出错就返回一个负数。

virtual int GetStringLen(int index) const;

返回 index 指定的项目的字符串长度，如果出错就返回一个负数。

int GetText(char far * str, int maxChars) const;

返回在编辑框或静态控件中的字符串。

int GetTextLen() const;

返回在编辑框或静态控件中的字符串长度。

void HideList();

隐藏组合框的列表区域。

virtual int InsertString(const char far * str, int index);

在 index 指定的项目前插入一个字符串。返回这个项目在列表中的索引，如果出错就返回一个负数。

int SetEditSel(int startPos, int endPos);

在组合框的编辑框中选择从 startPos 到 endPos 所指定的文本，如果组合框中没有编辑框就返回 CB_ERR。

int SetExtendedUI(bool extended);

如果组合框具有扩展用户界面，此函数就设置扩展用户界面。

virtual int SetItemData(int index, uint32 data);

设置列表框中的每一个项目的 32 位值，如果出错就返回一个负数。

int SetItemHeight(int index, int height);

设置项目的高度。

virtual int SetSelIndex(int index);

设置 index 指定的项目的状态为选择状态。如果 index 的值为 0，就选择列表中的第一个项目；如果 index 的值为－1，就放弃当前的选择；如果出错就返回一个负数。

virtual int SetSelString(const char far * findStr, int indexStart);

设置指定的字符串为选择状态，并将它拷贝到编辑框中。

void SetText(const char far * string);

选择以 string 字符串开头的项目。如果没有匹配的项目，SetText 就设置编辑框的内容为指定的字符串并选择它。

void ShowList();

void ShowList(bool show);

显示或隐藏列表。

uint Transfer(void * buffer, TTransferDirection direction);

传输组合框的项目和选择状态。buffer 指向一个 TComboBoxData 结构。如果 direction 是 tdGetData，数据从缓冲区中传出；如果 direction 是 tdSetData，数据从缓冲区中传入；Transfer 总是返回传输数据的大小。如果只想得到这个结构的大小而不想传入或传出数据，就设置 direction 的值为 tdSizeData。

Protected 成员函数：

virtual char far * GetClassName();

返回组合框注册类名“ComboBox”。

void SetupWindow();

配置窗口并限制用户可以在编辑框中虽然的字符串长度为 TextLen－1。

A.24　TComboBoxData 结构

头文件： combobox.h

描述： 负责传输 TComboBox 类数据到一个传输缓冲区的界面对象。

Public 构造函数和析构函数：

TComboBoxData();

~TComboBoxData();

Public 成员函数：

void AddString(const char * str, bool isSelected = false);

在字符串数组 Strings 中增加一个指定的字符串。如果 IsSelected 的值为 true，AddString 函数就清除字符串 Selection，并将指定的字符串拷贝到 Selection 中。

void AddStringItem(const char * str, uint32 itemData, bool isSelected = false);

调用 AddItemData 函数在 ItemDatas 数组中增加一个项目数据，调用 AddString 函数在 Strings 数组中增加一个字符串。

void Clear();

将字符串 Strings 和 ItemDatas 置空，重新设置索引与字符串的值。

TDwordArray& GetItemDatas();

从列表中返回一个 DWORD 数组。

int GetSelCount() const;

返回选择的项目个数，0 或 1。

string& GetSelection();

从列表中返回当前选择的字符串。

int GetSelIndex();

返回所选择的项目的索引。

void GetSelString(char far * buffer, int bufferSize) const;

将所选择的字符串拷贝到 buffer 中，bufferSize 确定 buffer 的大小。

int GetSelStringLength() const;

返回当前选择的字符串菜单。

TStringArray& GetStrings();

返回字符串数组。

void ResetSelections();

重新数组当前选择项目的索引和字符串。

void Select(int index);

按照 index 的值选择项目。

void SelectString(const char far * str);

按照 str 指定的字符串选择项目。

Protected 数据成员：

TDwordArray ItemDatas;

与列表框进行数据交换的 DWORD 类型数组。

string Selection;

当前所选择项目的字符串。

int SelIndex;

当前所选择项目的索引。

TStringArray Strings;

字符串数组。

A.25　TValidator 类

头文件：validate.h

描述：TValidator 类是可流类，定义了一个格式字符串用来规定数据的格式，它在 OWL 5.0 中的位置如图A－21所示。

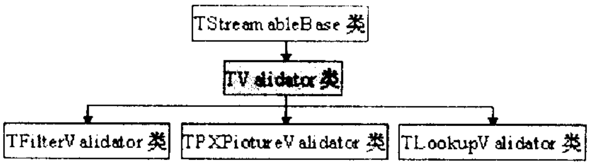


图 A－21　TValidator 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

TValidator();

virtual ～TValidator();

Public 成员函数：

virtual void Error();

当用户的输入为非法时调用。

bool HasOption(int option);

如果设置了 Options 参数就返回 true。

virtual bool IsValid(const char far * str);

如果数据合法就返回 true。

virtual bool IsValidInput(char far * str, bool suppressFill);

如果编辑控件中的用户输入合法就返回 true。

void SetOption(int option);

设置 Options 参数。

virtual uint Transfer(char far * str, void * buffer, TTransferDirection direction);

允许数据传输。

void UnsetOption(int option);

删除 Options 参数。

bool Valid(const char far * str);

如果 IsValid 返回 true，Valid 函数亦返回 true。

Protected 数据成员：

uint16 Options;

定义类型：

enum TValidatorOptions{voFill, voTransfer, voOnAppend, voReserved};

A.26　TFilterValidator 类

头文件：validate.h

描述：TFilterValidator 类是可流类，可以检查一个用户输入是否满足要求。TFilterValidator 类在 OWL 5.0 中的位置如图 A-22 所示。

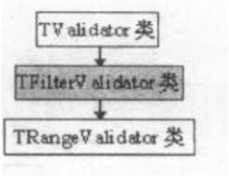


图 A-22　TFilterValidator 类在 OWL 5.0 中的位置

Public 构造函数：

TFilterValidator(const TCharSet& validChars);

Public 成员函数：

virtual void Error();

当用户的输入为非法时调用。

virtual bool IsValid(const char far * str);

如果数据合法就返回 true。

virtual bool IsValidInput(char far * str, bool suppressFill);

如果编辑控件中的用户输入合法就返回 true。

Protected 数据成员：

TCharSet ValidChars;

字符格式。

A.27　TRangeValidator 类

头文件：validate.h

描述：TRangeValidator 类是可流类，它确定用户的数字输入是否在规定的范围内。TRangeValidator 类在 OWL 5.0 中的位置如图 A-23 所示。

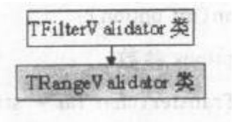


图 A-23　TRangeValidator 类在 OWL 5.0 中的位置

Public 构造函数：

TRangeValidator(long min, long max);

Public 成员函数：

virtual void Error();

当用户的输入为非法时调用。

virtual bool IsValid(const char far * str);

如果数据合法就返回 true。

uint Transfer(char far * str, void * buffer, TTransferDirection direction);

允许数据传输。

Protected 数据成员：

long Max;

允许范围的最大值。

long Min;

允许范围的最小值。

A.28　TPXPictureValidator 类

头文件: validate.h

描述: TPXPictureValidator 类提供数据输入的格式模板，它在 OWL 5.0 中的位置如图A-24所示。

Public 构造函数:

TPXPictureValidator(const char far * pic, bool autoFill=false);

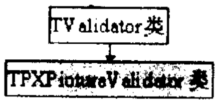


图 A-24　TPXPictureValidator 类在 OWL 5.0 中的位置

Public 成员函数:

void Error();

当用户的输入为非法时调用。

bool IsValid(const char far * str);

如果数据合法就返回 true。

bool IsValidInput(char far * str, bool suppressFill);

如果编辑控件中的用户输入合法就返回 true。

virtual TPicResult Picture(char far * input, bool autoFill=false);

获得输入检查的结果。

Protected 数据成员:

string Pic;

模板字符串。

A.29　TLookupValidator 类

头文件: validate.h

描述: TLookupValidator 是一个可流类，它将一个字符串与一个可接收的字符串进行比较。TLookupValidator 类在 OWL 5.0 中的位置如图 A-25 所示。

Public 构造函数:

TLookupValidator();

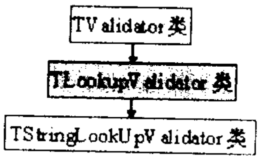


图 A-25　TLookupValidator 类在 OWL 5.0 中的位置

Public 成员函数:

bool IsValid(const char far * str);

如果输入合法就返回 true。

virtual bool Lookup(const char far * str);

如果在表中找到所给的字符串就返回 true。

A.30 TStringLookupValidator 类

头文件：validate.h

描述：作为 TLookupValidator 的派生类，TStringLookupValidator 也是一个可流类，用于检查用户的输入是否是规定的一系列字符串中的一个。TStingLookupValidator 在 OWL 5.0 中的位置如图 A-26 所示。

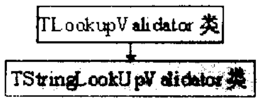


图 A-26 TStingLookupValidator 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

TStringLookupValidator(TSortedStringArray * strings);

~TStringLookupValidator();

Public 成员函数：

void Error();

当用户的输入为非法时调用。

bool Lookup(const char far * str);

如果在表中找到所给的字符串就返回 true。

void NewStringList(TSortedStringArray * strings);

设置新的字符串数组。

Protected 数据成员：

TSortedStringArray * Strings;

合法的字符串数组。

A.31 TDialog 类

头文件：dialog.h

描述：ObjectWindows 为对话框提供了 3-D 界面，如果你想要使用 Microsoft 公司的 CTL3D DLL，只要在应用程序中调用 TApplication::EnableCtl3d 即可，ObjectWindows 会自动发送 WM_CTLCOLOR 消息到 CTL3D DLL。

ObjectWindows 也为对话框提供了 BWCC 界面，缺省时 TDialog 类使用 BWCC。TDialog 类是个可流类，它在 OWL 5.0 中的位置如图 A-27 所示。

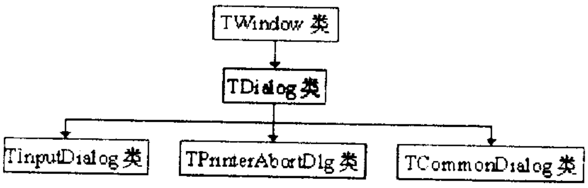


图 A-27 TDialog 类在 OWL 5.0 中的位置

Public 数据成员：

TDialogAttr Attr;

创建对话框对象的大小和风格参数。

bool IsModal;

如果是模式对话框，IsModal 的值为 true；如果是无模式对话框，值为 false。

Public 构造函数和析构函数：

```
TDialog(TWindow * parent,
        TResId resId,
        TModule * module = 0);
~TDialog();
```

Public 数据成员：

void CloseWindow(int retValue = IDCANCEL);

重载 TWindow 类中定义的同名函数，关闭对话框。如果是无模式对话框，调用 TWindow::CloseWindow；如果是模式对话框，调用 CanClose。如果 CanClose 函数返回 true，CloseWindow 函数调用 TransferData 函数传输参数 retValue 的值。retValue 的缺省值是 IDCANCEL。

void CmCancel();

用户按 Cancel 后的自动响应，调用 Destroy 函数传输 IDCANCEL 值。

void CmOk();

用户按 OK 后的自动响应，调用 CloseWindow 函数传输 IDOK 值。

virtual bool Create();

创建一个无模式对话框的界面元素。

virtual void Destroy(int retValue = IDCANCEL);

删除 TDialog 对象的界面元素。如果是无模式对话框，Destroy 调用 TWindow::Destroy 函数；如果是模式对话框，Destroy 在全部的子窗口中调用 EnableAutoCreate 函数，然后调用 EndDialog 函数传输 retValue 值。retValue 的缺省值是 IDCANCEL。

virtual bool DialogFunction(uint message, WPARAM wParam, LPARAM lParam);

想要通过对话框函数处理消息，你需要重载这个函数。如果消息被处理了，DialogFunction 函数返回 true，否则返回 false。

virtual HWND DoCreate();

DoCreate 函数由 Create 函数调用，用来创建一个无模式对话框。

virtual int DoExecute();

DoExecute 函数由 Execute 函数调用，用来执行一个模式对话框。

void EvClose();

响应一个关闭窗口时输入的 EvClose 消息。

HBRUSH EvCtlColor(HDC hDC, HWND hWndChild, uint ctlType);

传输子窗口的设备上下文句柄、子窗口句柄和缺省的系统颜色到父窗口中，父窗口通过所给的设备上下文句柄 hDC 来设置子窗口的文本和背景颜色。如果 3D 功能开启，EvCtlColor 通过 CTL3D DLL 处理 EV_WM_CTLCOLOR 消息。

virtual bool EvInitDialog(HWND hWndFocus);

对话框显示之前自动调用 EvInitDialog 函数，它又调用 SetupWindow 来设置对话框和其上的控件。

void EvPaint();

EvPaint 函数调用 TWindow 类的 DefaultProcessing 函数来重画窗口。

virtual void EvSetFont(HFONT hfont, bool redraw);

关闭窗口字体时作出响应。

virtual int Execute();

创建并执行一个模式对话框，如果创建成功，Execute 要到对话框关闭后才返回。函数的执行过程为

(1) 登记对话框窗口类和所有对话框的子窗口；

(2) 调用 DoExecute 来执行对话框；

(3) 如果出错，进行处理。

Execute 函数返回一个整数值代表对话框关闭的方式。如果对话框没有被创建，返回 -1。

uint GetDefaultId() const;

获得缺省的 ID 值。

HWND GetItemHandle(int childId);

返回 childId 指定的对话框窗口句柄。因为 GetItemHandle 已经过时了，新的应用程序一般使用 TWindow::GetDlgItem 函数。

bool PerformDlgInit();

以对话框资源 ID 值 RT_DLGINIT 中的内容初始化对话框，如果成功返回 true，否则返回 false。

bool PreProcessMsg(MSG& msg);

重载 TWindow 类的同名函数，预先处理窗口消息。

uint32 SendDlgItemMsg(int ChildId, uint16 Msg, uint16 WParam, uint32 LParam);

发送一个 Msg 指定的控件消息到 ChildID 值指定的控件对象，WParam 和 LParam 是控件消息中的信息。SendDlgItemMsg 返回控件返回的值，如果 ID 值非法，则返回 0(这个函数过时了)。

void SetCaption(const char far * title);

用 title 参数所指的字符串设置对话框的标题。

void SetDefaultId(uint Id);

设置缺省的 ID 值。

Protected 成员函数：

char far * GetClassName();

重载 TWindow 类的同名函数，返回对话框的缺省窗口类名(只用于无模式对话框)。对于模式对话框，GetClassName 函数返回缺省的类名。如果 BWCC 开启，GetClassName 函数返回 BORDLGCLASS。

void GetWindowClass(WNDCLASS& wndClass);

重载 TWindow 类的同名函数，用一个已有的登记参数(如果 BWCC 开启就使用 BWCC 的登记参数)来填充 WndClass。

void SetupWindow();

重载 TWindow 类的同名函数，调用 SetCaption (设置标题)和 TWindow::SetupWindow 函数设置对话框。如果 3D 开启，SetupWindow 函数就调用 CTL3D DLL。

事件响应表：见表 A-37

表 A-37

EV_ macros	成员函数
EV_COMMAND (IDCANCEL, CmCancel)	CmCancel
EV_COMMAND (IDOK, CmOk)	CmOk
EV_WM_CTLCOLOR	EvCtlColor
EV_WM_CLOSE	EvClose
EV_WM_PAINT	EvPaint

定义结构：

TDialog class::TDialogAttr struct

struct TDialogAttr {char far * Name;uint32 Param};

A.32　TInputDialog 类

头文件：inputdia.h

描述：提供一个供用户输入用的对话框，当输入对话框创建的时候，其标题条、说明文字和缺省的编辑框亦被同时创建。TInputDialog 是一个可流类，它在 OWL 5.0 中的位置如图 A-28 所示。

Public 数据成员：

char far * buffer;

指向传输输入文本的缓冲区。在输入对话框的构造函数中作为参数时，包含了编辑框中的缺省文本。

int BufferSize;

用户输入文本缓冲区的大小。

char far * prompt;

指向输入对话框的说明文字。

Public 构造函数：

```
TInputDialog(TWindow * parent,
             const char far * title,
             const char far * prompt,
             char far * buffer,
             int buffersize,
             TModule * module = 0,
             TValidator * valid = 0)
```

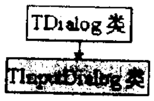


图 A-28　TinputDialog 类在 OWL 5.0 中的位置

Public 成员函数：

void TransferData(TTransferDirection direction)

传输输入对话框的数据。如果 direction 的值是 tdSetData，设置对话框的静态控件和编辑框的值为传输缓冲区中的值；如果 direction 的值是 tdGetData，用当前的编辑框中的文本填充传输缓冲区。

Protected 成员函数：

virtual void SetupWindow();

调用 TDialog::SetupWindow 函数来设置窗口，然后限制用户可以输入的字符串长度为 bufferSize − 1。

A.33　TPrinterAbortDlg 类

头文件：printer.h

描述：TPrinterAbortDlg 是缺省的打印退出对话框。这个对话框初始化后显示正在打印的文件名、打印设备名和打印所用的接口。

TPrinterAbortDlg 包含了三个静态文本控件，标题的 ID 值为 101，打印设备的 ID 值为 102，输出端口的 ID 值为 103。在字符串中必须加上"%s"，以便被标题、打印设备和输出端口的字符串取代。TPrinterAbortDlg 类在 OWL 5.0 中的位置如图 A－29 所示。

Public 构造函数：

```
TPrinterAbortDlg(TWindow * parent,
                 TResId resId,
                 const char far * title,
                 const char far * device,
                 const char far * port);
```

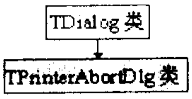


图 A－29　TPrinterAbortDlg 类在 OWL 5.0 中的位置

Protected 成员函数：

virtual LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);

处理打印退出对话框中的 Cancel 按钮事件。

void SetupWindow();

把对话框资源与对话框标题、打印设备和输出端口联系起来。

A.34　TCommonDialog 类

头文件： commdial.h

描述： TCommonDialog 是从 TDialog 类中派生出来，是通用对话框对象的虚基类，提供了创建通用对话框的基本功能。ObjectWindows 中从这个类派生出来的类有：

* TChooseColorDialog　　让用户选择颜色的模式对话框；
* TChooseFontDialog　　让用户选择字体的模式对话框；
* TReplaceDialog　　让用户取代文字的无模式对话框；
* TFindDialog　　让用户寻找文字的无模式对话框；
* TFileOpenDialog　　让用户打开特定文件的模式对话框；
* TFileSaveDialog　　让用户以特定文件存盘的模式对话框；
* TPrintDialog　　让用户进行打印设置的模式对话框。

每个通用对话框类都有一个伴随的数据类 TData，它存储着对话框特性参数和用户的使用数据。比如，TChooseColorDialog::TData 类保存了用户选择的颜色参数。事实上，在构造一个通用对话框对象之前就构造了一个 TData 对象，填充 TData 对象的数据成员。然后执行一个模式对话框或创建一个无模式对话框。

TCommonDialog 类在 OWL 5.0 中的位置如图 A－30 所示。

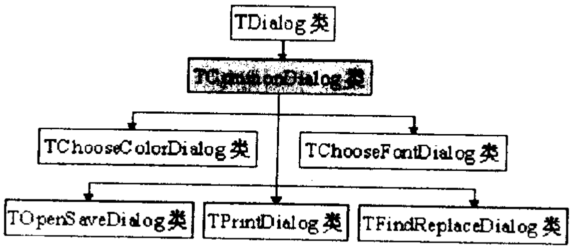


图 A－30　TCommonDialog 类在 OWL 5.0 中的位置

Public 构造函数：

```
TCommonDialog(TWindow * parent,
              const char far * title = 0,
              TModule * module = 0);
```

Public 成员函数：

HWND DoCreate();

DoCreate 函数被 Create 函数调用，用于创建一个无模式对话框。如果不成功则返回 0。

int DoExecute();

DoExecute 被 Execute 调用，用于执行一个模式对话框。如果对话框执行不成功或按 cancel 按钮退出，返回 IDCANCEL。

Protected 数据成员：

const char far * CDTitle;

CDTitle 存储了显示在通用对话框上的标题。

Protected 成员函数：

void CmHelp();

处理在对话框中按 Help 按钮事件。

void CmOkCancel();

响应在对话框中按 OK 和 Cancel 按钮事件，调用 DefaultProcessing 函数处理事件。

void EvClose();

响应 WM_CLOSE 消息，调用 DefaultProcessing 函数处理事件。

void SetupWindow();

分配对话框的标题到 CDTitle 中。

事件响应表：见表 A－38

表 A－38

EV_ macros	成员函数
EV_COMMAND(IDCANCEL, CmOkCancel)	CmOkCancel
EV_COMMAND(IDOK, CmOkCancel)	CmOkCancel
EV_WM_CLOSE	EvClose
EV_WM_CTLCOLOR	EvCtlColor

A.35　TChooseColorDialog 类

头文件：chooseco.h

描述：TChooseColorDialog 类是一个模式对话框，它允许用户选择调配颜色。TChooseColorDialog 类可以无模式方式显示，把对话框的父窗口作为一个不可见的弹出窗口，将这个弹出窗口作为应用程序主窗口的子窗口。

TChooseColorDialog 类使用 TChooseColor::TData 结构来初始化用户的颜色选择，它在 OWL 5.0 中的位置如图 A-31 所示。

Public 构造函数：

```
TChooseColorDialog(TWindow * parent,
                   TData& data,
                   TResId templateID = 0,
                   const char far * title = 0,
                   TModule * module = 0);
```

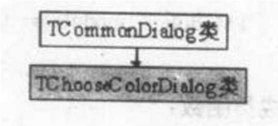


图 A-31　TChooseColorDialog 类在 OWL 5.0 中的位置

Public 成员函数：

void SetRGBColor(TColor color);

发出一个 SetRGBMsgId 消息来设置当前的 RGB 颜色。

Public 数据成员：

CHOOSECOLOR cc;

存储 TChooseColorDialog 结构的长度、拥有此对话框的窗口和对话框样板的数据块，它也指向包含 16 个 RGB 值的数组，这些 RGB 值出现在对话框的用户颜色框中。数组中还包括对话框的初始化标志。

TData& Data;

Data 作为构造 TData 对象时的参数。

static uint SetRGBMsgId;

SetRGBColor 发出的注册消息的 ID 值。

Protected 成员函数：

bool DialogFunction(uint message, WPARAM, LPARAM);

如果事件处理了，返回 true。

int DoExecute();

如果没有错误发生，DoExecute 将标志和颜色拷贝到 Data 中并返回 0；如果发生错误，DoExecute 返回 IDCANCEL。Error 的值被设置为 CommDlgExtendedError 的返回值。

LPARAM EvSetRGBColor(WPARAM, LPARAM);

响应 SetRGBColor 发出的消息。这个事件处理没有包括在事件响应表中。

A.36　TChooseColorDialog::TData 类

头文件： chooseco.h

描述： 定义了初始化颜色选择对话框的必要信息。

Public 数据成员：

TColor Color;

对话框创建时颜色选择的初始值，包含了对话框改变时用户所选择的颜色。

TColor * CustColors;

指向一个包含 16 种颜色的数组。

uint32 Error;

如果对话框成功执行，其为0；其它情况下的值及意义见表A-39。

表A-39

取　　值	意　　义
CDERR_DIALOGFAILURE	创建对话框失败
CDERR_FINDRESFAILURE	没有找到特定的资源
CDERR_LOADRESFAILURE	装入资源失败
CDERR_LOCKRESOURCEFAILURE	锁定特定资源失败
CDERR_LOADSTRFAILURE	装入特定字符串失败

uint32 Flags;

控制对话框的型式和功能的标志，可以是表A-40所列值的组合。

表A-40

取　值	意　　义
CC_FULLOPEN	当对话框创建时显示全部的对话框
CC_PREVENTFULLOPEN	关闭"Define Custom Colors"按钮
CC_RGBINIT	使对话框使用 rgbResult 中定义的颜色作为初始颜色选择
CC_SHOWHELP	使对话框显示 Help 按钮

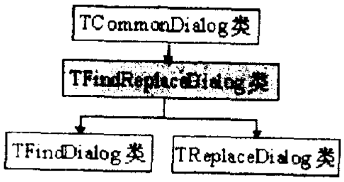
A.37 TFindReplaceDialog 类

头文件：findrepl.h

描述：TFindReplaceDialog 类是一个抽象类，定义了一个无模式对话框供用户查找和替换文本。这个基类的功能适用于它的两个派生类，即 TFindDialog 类(查找文本)和 TReplaceDialog(替换文本)。TFindReplaceDialog 提供一个注册的消息与父窗口联系，它在 OWL 5.0 中的位置如图A-32所示。

Public 构造函数：

```
TFindReplaceDialog(TWindow * parent,
        TData& data,
        TResId templateId = 0,
        const char far * title = 0,
        TModule * module = 0);
```



图A-32 TFindReplaceDialog 类在 OWL 5.0 中的位置

Public 成员函数：

void CmCancel();= 0, TModule * module = 0);

响应用户按 Cancel 按钮事件。

void CmFindNext();

响应用户按 Find 按钮事件。

void CmReplace();

响应用户按 Replace 按钮事件。

void CmReplaceAll();

响应用户按 ReplaceAll 按钮事件。

void EvNCDestroy();

调用 TWindow::EvNCDestroy 函数响应一个 EV_WM_NCDESTROY 消息输入。

Protected 数据成员：

TData& Data;

Data 作为构造 TData 对象时的参数。

FINDREPLACE fr;

包含查找和替换特性(比如，查找缓冲区的大小和查找替换字符串的指针)的结构。

Protected 成员函数：

HWND DoCreate()=0;

DoCreate 是一个虚函数，它在派生类中将被重载，用来创建一个无模式的查找和替换对话框。

bool DialogFunction(uint message, WPARAM, LPARAM);

事件处理函数，如果事件处理了，就返回 true。

void Init(TResId templateId);

在派生类的构造函数中被调用，根据资源 ID 值和其他数据成员初始化一个 TFindReplaceDialog 对象。

事件响应表：见表 A-41

表 A-41

EV_ macros	成员函数
EV_WM_NCDESTROY	EvNCDestroy

A.38 TFindReplaceDialog::TData 类

头文件：findrepl.h

描述：TFindReplaceDialog::TData 类封装了初始化一个 TFindReplace 对话框必要的信息。TFindDialog 类和 TReplaceDialog 类使用 TFindReplaceDialog::TData 类来初始化对话框并接收用户的输入，比如查找和替代的字符串。

Public 构造函数和析构函数

TFindReplaceDialog::TData(uint32 flags = 0, int bufferSize = 81);

~TData();

Public 数据成员：

int BuffSize;

缓冲区的大小。

uint32 Error;

如果对话框成功执行，值为 0；其它情况下的值及意义见表 A-42。

表 A-42

取　　值	意　　义
CDERR_LOADRESFAILURE	装入资源失败
CDERR_LOCKRESOURCEFAILURE	锁定特定资源失败
CDERR_REGISTERMSGFAIL	一个用于应用程序之间联系的窗口消息没有注册
CDERR_LOADSTRFAILURE	装入特定字符串失败

char * FindWhat;

待查找的字符串。

char * ReplaceWith;

要替换的字符串。

uint32 Flags;

Flags 描述对话框中按钮的状态和动作，可以是表 A-43 所列值的组合。

表 A-43

取　　值	意　　义
FR_DOWN	在查找对话框的方向组中选择 Down 按钮
FR_HIDEMATCHCASE	Match Case 检查框被隐藏
FR_HIDEWHOLEWORD	Whole Word 检查框被隐藏
FR_HIDEUPDOWN	Up 和 Down 按钮被隐藏
FR_MATCHCASE	Match Case 检查框为选择状态
FR_NOMATCHCASE	Match Case 检查框为非选择状态
FR_NOUPDOWN	Up 和 Down 按钮被禁止使用
FR_NOWHOLEWORD	Whole Word 检查框被禁止使用
FR_REPLACE	Replace 按钮在替代对话框中使用
FR_REPLACEALL	Replace All 按钮在替代对话框中使用
FR_WHOLEWORD	Whole Word 检查框的状态为选择状态

A.39　TFindDialog 类

头文件：findrepl.h

描述：TFindDialog 类创建一个无模式对话框供用户查找文本，它在 OWL 5.0 中的位置如图 A-33 所示。TFindDialog 提供一个注册的消息与父窗口联系。因为是从 TFindReplaceDialog 类派生出来的，TFindDialog 使用结构 TFindReplaceDialog::TData 并根据用户输入的值(比如，要查找的文本)来初始化对话框。

Public 构造函数：

TFindDialog(TWindow * parent,

```
            TData& data,
            TResId templateId = 0,
            const char far * title = 0,
            TModule * module=0);
```

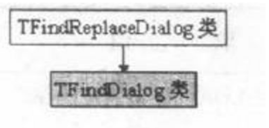


图 A - 33　TFindDialog 类在 OWL 5.0 中的位置

Protected 成员函数：

HWND DoCreate();

创建一个无模式查找对话框。

A. 40　TReplaceDialog 类

头文件： findrepl. h

描述： TReplaceDialog 类创建一个无模式对话框供用户替换文本，它在 OWL 5.0 中的位置如图 A - 34 所示。由于是无模式对话框，用户可以一边编辑文本一边进行文本替换。另外，由于 TReplaceDialog 是从 TFindReplaceDialog 类派生出来的，故使用结构 TFindReplaceDialog::TData 并根据用户输入的值(比如，要替换的文本)来初始化对话框。

Public 构造函数：

```
TReplaceDialog(TWindow * parent,
            TData& data,
            TResID templateName=0,
            const char far * title=0,
            TModule * module=0);
```

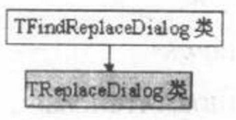


图 A - 34　TReplaceDialog 类在 OWL 5.0 中的位置

Protected 成员函数：

HWND DoCreate();

创建一个无模式替换对话框。

A. 41　TChooseFontDialog 类

头文件： choosefo. h

描述： TChooseFontDialog 对象是模式对话框，供用户选择字体、字体风格(比如加粗或倾斜)、字体大小、效果(比如删除线或下划线)和字体颜色。TChooseFontDialog 对象可作为无模式对话框显示，它把对话框的父窗口作为一个不可见的弹出窗口，再将这个弹出窗口作为应用程序主窗口的子窗口。TChooseFontDialog 使用 TChooseFontDialog::TData 结构来初始化对话框的用户选择字体，它在 OWL 5.0 中的位置如图 A - 35 所示。

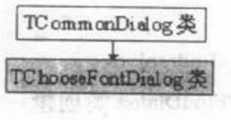


图 A - 35　TChooseFontDialog 类在 OWL 5.0 中的位置

Public 构造函数：

```
TChooseFontDialog(TWindow * parent,
            TData& data,
            TResID templateID = 0,
```

const char far * title = 0,
TModule * module = 0);

Protected 数据成员：

CHOOSEFONT cf;

字体特性。cf 初始化为 TChooseFontDialog::TData 结构中的值，它存储结构的长度、拥有此对话框的窗口和包含对话框样板的数据块，还包括了对话框初始化标志。

TDaa& Data;

Data 作为构造 TData 对象时的参数。

Protected 成员函数：

void CmFontApply();

处理 FontApply 按钮按下的事件。

bool DialogFunction(uint message, WPARAM, LPARAM);

如果事件处理了，返回 true。

int DoExecute();

如果没有错误，DoExecute 将标志 flag 的值和字体信息拷贝到 Ddata 中，返回 IDOK 或 IDCANCEL。如果发生错误，返回 TChooseFontDialog::TData 结构中的错误代码。

A.42 TChooseFontDialog::TData 类

头文件：choosefo.h

描述：定义初始化字体选择对话框的必要信息。

Public 数据成员：

TColor Color;

设置对话框创建时字体颜色选择的初始值;包含了对话框改变时用户所选择的字体颜色。

HDC DC;

字体获得的设备上下文句柄。

uint32 Error;

如果对话框成功执行，值为 0，其它情况下的值及意义见表 A-44。

表 A-44

取　　值	意　　义
CDERR_DIALOGFAILURE	创建对话框失败
CDERR_FINDRESFAILURE	没有找到特定的资源
CDERR_LOADRESFAILURE	装入资源失败
CDERR_LOCKRESOURCEFAILURE	锁定特定资源失败
CFERR_MAXLESSTHANMIN	SizeMax 中的定义值小于 SizeMin 中的定义值
CFERR_NOFONTS	没有字体存在
CDERR_LOADSTRFAILURE	装入特定字符串失败

uint32 Flags;

　　控制对话框的型式和功能的标志，可以是表 A - 45 所列值的组合。

uint16 FontType;

　　字体类型或名字。

LOGFONT LogFont;

　　字体风格。

int PointSize;

　　字体大小。

int SizeMax;

　　字体允许的最大尺寸。

表 A - 45

取　值	意　　义
CF_APPLY	打开并显示 Apply 按钮
CF_ANSIONLY	设置 ChooseFontDialog 结构只使用 ANSI 字符集的字体
CF_BOTH	设置对话框显示可用的打印机字体和屏幕字体
CF_EFFECTS	打开 strikeout、underline 和颜色效果功能
CF_FIXEDPITCHONLY	只打开 fixed—pitch 字体
CF_FORCEFONTEXIST	如果用户选择了一个不存在的字体就显示出错信息
CF_INITTOLOGFONTSTRUCT	LogFont 指向对话框初始化时使用 LOGFONT 结构
CF_LIMITSIZE	将字体选择限制在 SizeMin 和 SizeMax 之间
CF_NOSIMULATIONS	不允许 GDI 字体仿真
CF_PRINTERFONTS	对话框只显示打印机要求的字体
CF_SCALABLEONLY	只允许选择 scalable 字体
CF_SCREENFONTS	只显示系统提供的屏幕字体
CF_TTONLY	只显示和允许选择 TrueType 字体
CF_USESTYLE	按照一个缓冲区中定义的字体风格设置字体
CF_WYSIWYG	只允许选择打印机和屏幕可用的字体
CC_SHOWHELP	使对话框显示 Help 按钮

int SizeMin;

　　字体允许的最小尺寸。

char far * Style;

　　字体风格，比如，bold、italic、underline 或 strikeout。

A.43　TPrintDialog 类

头文件：printdia.h

描述：TPrintDialog 显示一个模式打印或一个打印设置对话框。打印对话框供用户设置打印任务的特性；打印设置对话框则供用户配置打印机和设置扩展打印任务特性。你也可以使用 TPrinter 和 TPrintout 来提供打印机对话框。TPrintDialog 使用 TPrintDialog::TData 结构来初始化用户的打印机设置对话框，比如，打印页数、输出设备等等，它在 OWL 5.0 中的位置如图 A-36 所示。

Public 构造函数：

```
TPrintDialog(TWindow * parent,
             TData& data,
             const char far * printTemplateName=0,
             const char far * setupTemplateName=0,
             const char far * title=0,
             TModule * module=0);
```

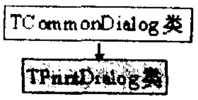


图 A-36　TPrintDialog 类在 OWL 5.0 中的位置

Public 成员函数：

int DoExecute();

如果没有错误，DoExecute 将 flags 和打印设置拷贝到构造函数的 data 参数中；如果发生错误，DoExecute 将错误号设置为 TPrintDialog::TData::Error 定义的号码。

bool GetDefaultPrinter();

不显示对话框，GetDefaultPrinter 获得系统缺省打印设备模式和名字。

Protected 数据成员：

TData& Data;

Data 作为构造 TData 对象时的参数。TData 对象包括了打印设置，比如，打印的份数、打印页数以及输出设备名等等。

PRINTDLG pd;

对话框打印任务特性，比如，打印页区域、打印份数、设备上下文和其它一个初始化打印对话框或打印设置对话框所必须的信息。

Protected 成员函数：

void CmSetup();

响应伴随 EV_COMMAND 消息的按下 setup 按钮事件。

bool DialogFunction(uint message, WPARAM, LPARAM);

如果消息被处理则返回 true。

A.44　TOpenSaveDialog 类

头文件：opensave.h

描述：TOpenSaveDialog 是一个供用户以特定文件名打开和保存文件的对话框的基类。TOpenSaveDialog

构造了一个结构 TData，将它传输给 TOpenSaveDialog 的构造函数，然后就以模式或无模式对话框的方式创建对话框。TOPENsAWEdIALOG 类在 OWL 5.0 中的位置如图 A-37 所示。

Public 构造函数：

```
TOpenSaveDialog(TWindow * Parent,
                TData& data,
                TResID templateID = 0,
                const char far * title = 0,
                TModule * module = 0);
```

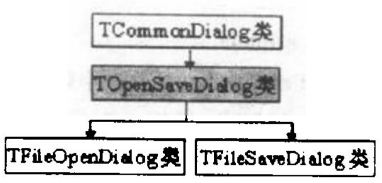


图 A-37 TOpenSaveDialog 类在 OWL 5.0 中的位置

Public 成员函数：

static int GetFileTitle(const char far * fileName, char far * fileTitle, int fileTitleLen);

static int GetFileTitleLen(const char far * fileName);

Protected 数据成员：

TData& Data;

保存了文件名、文件长度、扩展名、可选择的文件类型、初始路径、缺省文件名、扩展名以及全部的出错信息。

OPENFILENAME ofn;

文件的属性，比如，长度、扩展名和路径。ofn 用 TOpenSaveDialog::TData 中定义的数据初始化。

static uint ShareViMsgId;

注册 ShareViolation 消息的消息 ID 值。

Protected 构造函数：

TOpenSaveDialog(TWindow * Parent, TData& data, TModule * module);

Protected 成员函数：

void CmLbSelChanged();

描述文件名列表框的选择状态。

void CmOk();

响应 OK 按钮按下的事件。

bool DialogFunction(uint message, WPARAM, LPARAM);

如果消息被处理则返回 true；如果发生了一个文件共享错误，返回 ShareViMsgId；否则返回 false。

int DoExecute();

创建或执行对话框。

void Init(TResId templateId);

以当前的资源 ID 值初始化 TOpenSaveDialog 对象。

virtual int ShareViolation();

当发生文件共享错误时调用。

A.45 TOpenSaveDialog::TData 结构

头文件： opensave.h

描述：TOpenSaveDialog 的结构包括了用户打开或保存的文件选择信息，保存了文件名、文件长度、扩展名、可选择的文件类型、初始路径、缺省文件名、扩展名以及全部的出错信息。TFileOpenDialog 类和 TFileSaveDialog 类在打开和保存文件时使用保存在这个结构中的信息。

数据成员：

char * CustomFilter;

CustomFilter 保存了用户定义的可选文件类型，比如 *.txt 或 *.cpp。

char * DefExt;

保存文件的扩展名。

uint32 Error;

如果对话框成功执行，值为 0；其余情况下的值及意义见表 A-46。

表 A-46

取 值	意 义
CDERR_LOADRESFAILURE	装入资源失败
CDERR_LOCKRESOURCEFAILURE	锁定特定资源失败
CDERR_DIALOGFAILURE	对话框创建失败
CDERR_LOADSTRFAILURE	装入特定字符串失败

uint32 Flags;

Flag 的值可以是表 A-47 所列值的组合：

表 A-47

取 值	意 义
OFN_HIDEREADONLY	隐藏只读检查框
OFN_FILEMUSTEXIST	让用户只选择在文件列表框中的文件，否则提示错误
OFN_PATHMUSTEXIST	让用户只能输入正确的路径，否则提示错误
OFN_NOVALIDATE	不检查文件名
OFN_NOCHANGEDIR	设置对话框创建时的当前路径
OFN_ALLOWMULTISELECT	在文件列表框中允许多个选择
OFN_CREATEPROMPT	当用户创建一个新文件时给出提问
OFN_EXTENSIONDIFFERENT	表示用户输入的文件名的扩展名与 DefExt 要求的不同，将这个消息返回调用者
OFN_NOREADONLYRETURN	返回的文件没有只读的属性并且路径没有写保护，将这个消息返回调用者
OFN_NOTESTFILECREATE	对话框关闭后创建文件
OFN_OVERWRITEPROMPT	Save As 对话框显示提问是否覆盖一个已存在的文件
OFN_SHAREAWARE	如果发生文件共享问题，返回所给的文件名
OFN_SHOWHELP	在对话框上显示 Help 按钮

char * FileName;

打开或保存的文件名。

char * Filter;

Filter 是定义的可选文件类型。

int FilterIndex;

可选文件类型的索引。

char * InitialDir;

初始化时的路径。

Public 成员函数：

void SetFilter(const char * filter = 0);

A.46　TFileOpenDialog 类

头文件：opensave.h

描述：TFileOpenDialog 是一个让用户打开特定文件的模式对话框，该对话框用于响应用户选择 File|Open 菜单项时发出的 CM_FILEOPEN 命令，TFileOpenDialog 类使用 TOpenSave::TData 结构来初始化文件打开对话框，它在 OWL 5.0 中的位置如图 A-38 所示。

Public 构造函数：

```
TFileOpenDialog(TWindow * parent,
            TData& data,
            TResID templateID = 0,
            const char far * title,
            TModule * module = 0);
```

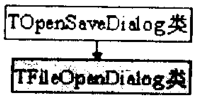


图 A-38　TFileOpenDialog 类在 OWL 5.0 中的位置

Public 成员函数：

int DoExecute();= 0

创建文件打开对话框。

A.47　TFileSaveDialog 类

头文件：opensave.h

描述：TFileSaveDialog 是一个让用户以特定文件名存盘的模式对话框，用于响应用户选择 File|Save 菜单项时发出的 CM_FILESAVEAS 命令。TFileSaveDialog 类使用 TOpenSave::TData 结构来初始化文件存盘对话框，它在 OWL 5.0 中的位置如图 A-39 所示。

Public 构造函数：

```
TFileSaveDialog(TWindow * parent,
            TData& data,
            TResID templateID = 0,
            const char far * title = 0,
            TModule * module = 0);
```

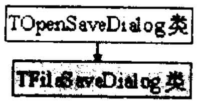


图 A-39　TFileSaveDialog 类在 OWL 5.0 中的位置

Public 成员函数：

int DoExecute()；

创建文件存盘对话框。

A.48　TGdiObject 类

头文件：gdiobjec.h

描述：TGdiObject 是 ObjectWindows 中的图形设备接口的基础类，封装了 Windows 中 GDI 的基本行为特征。用基于类的对象进行编程比通常的 GDI 编程更加直观并容易理解，TGdiOject 对象提供除了提供一个通常的 GDI 句柄以外还提供一个 C++对象的句柄。

作为一般 GDI 对象的基类，TGdiObject 类的派生类包括：画笔类 TPen、字体类 TFont、刷子类 TBrush、调色板类 TPalette 和位图类 TBitmap，它们都是设备上下文可以使用的对象，然而设备上下文类 TDC 却不是 TGdiObject 类的派生类。

TGdiObject 类中包括 GDI 句柄和一个 ShouldDelete 的标志，这个标志表明 GDI 句柄的删除情况。TGdiObject 类在 OWL 5.0 中的位置如图 A-40 所示。

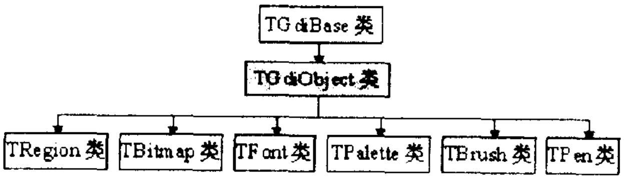


图 A-40　TGdiObject 类在 OWL 5.0 中的位置

Public 析构函数：

~TGdiObject()；

Public 枚举类型：

enum TAutoDelete{NoAutoDelete, AutoDelete}；

定义 GDI 句柄的标志。

enum TType{None, Pen, Brush, Font, Palette, Bitmap, TextBrush}；

枚举了可能的 GDI 对象类型。

Public 成员函数：

int GetObject(int count, void far * object) const；

获得 GDI 对象的信息。如果成功并且对象指针不是 0，函数就返回拷贝到对象缓冲区中的字节数；如果调用成功而对象指针的值为 0，函数就返回回该在缓冲区中存储对象所需的字节数。根据 GDI 对象的不同，函数的 object 可以得到 LOGPEN、LOGBRUSH、LOGFONT 或 BITMAP 结构的指针。

static void RefAdd(HANDLE handle, TType type)；

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。RefAdd 函数根据所给的句柄和类型为对象在对象信息表 ObjInfoBag 中增加一个参考入口并设置参考的值为 1。如果在表中已有对应的入口，函数就不进行任何操作。

static int RefCount(HANDLE handle)；

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。RefCount 函数返回所指对象的当前参考数目．如果在对象信息表 ObjInfoBag 中没有此对象，就返回－1。

static void RefDec(HANDLE handle);

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。RefDec 函数每次将对象的参考数目减1，直到这个对象已经没有参考时就删除这个对象。

static void RefDec(HANDLE handle, bool wantDelete);

当定义了__TRACE 时可以使用此函数。提供定义 wantDelete 的值，可以改变通常的删除方式。

static TObjInfo * RefFind(HANDLE object);

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。通过此函数可以查找对象的类型和参考数目，函数的返回值是 TObjInfo 类型的对象。如果没有找到匹配的对象就返回 0。

static void RefInc(HANDLE handle);

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。根据所给的对象句柄给对象的参考数目加 1。

static void RefRemove(HANDLE handle);

当没有定义 NO_GDI_ORPHAN_CONTROL 时可以调用此函数。RefRemove 函数删除所给对象在对象信息表 ObjInfoBag 中的参考入口；如果没有找到对象，函数就不进行任何操作。

operator HGDIOBJ()const

将 GDI 对象句柄转换为 HGDIOBJ 句柄。

Protected 数据成员：

HANDLE Handle;

指向对象的 GDI 句柄。

bool ShouldDelete;

如果析构函数需要删除对象，它的值就为 true。

Protected 成员函数：

void CheckValid(uint resId=IDS_GDIFAILURE);

static void CheckValid(HANDLE handle, uint resId=IDS_GDIFAILURE);

这两个版本的 CheckValid 函数的作用都是查找有效的 GDI 对象指针。

Protected 构造函数：

TGdiObject();

TGdiObject(HANDLE handle, TAutoDelete autoDelete = NoAutoDelete);

宏定义：

OBJ_REF_ADD(handle, type)

OBJ_REF_COUNT(handle)

OBJ_REF_DEC(handle, wantDelete)

OBJ_REF_INC(handle)

OBJ_REF_REMOVE(handle)

A.19　TRegion 类

头文件：gdiobjec.h

描述：TRegion 类是 TGdiObject 类的派生类，代表 GDI 抽象的形状和区域。TRegion 对象可以创建多变的区域对象，它在 OWL 5.0 中的位置如图 A-41 所示。

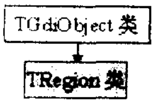


图 A-41　TRegion 类在 OWL 5.0 中的位置

类型定义：

enum TEllipse{Ellipse};

定义椭圆。

Public 构造函数：

TRegion();

缺省的构造函数，可以构造出一个空区域的对象。Handle 的值为 0，ShouldDelete 的值为 true。

TRegion(HRGN handle,TAutoDelete autoDelete = NoAutoDelete);

根据所给的句柄创建一个区域对象。Handle 的值就是所给的句柄，ShouldDelete 的值为 false。注意：在 C++对象删除的时候 GDI 句柄没有被删除。

TRegion(const TRegion& region);

这个拷贝构造函数创建一个所给对象的拷贝。

TRegion(const TRect& rect);

用所给的矩形区域创建区域对象。

TRegion(const TRect& E, TEllipse);

根据所给的椭圆区域构造一个区域对象。

TRegion(const TRect& rect, const TSize& corner);

用所给的矩形角构造一个区域对象。

TRegion(const TPoint * points,int count,int fillMode);

根据所给的点构成的多边形和填充模式构造一个区域对象。

TRegion(const TPoint * points, const int * polyCounts, int count,int fillMode);

创建一个填充的区域对象。

Public 成员函数：

bool Contains(const TPoint& point) const;

如果所给的点在区域对象中就返回 true。

int GetRgnBox(TRect& box) const;

找到包含区域的最小矩形。返回值和它的意义如表 A-48 所示。

表 A-48

取　值	意　义
COMPLEXREGION	区域具有覆盖边框
NULLREGION	区域为空
SIMPLEREGION	区域没有覆盖边框

TRect GetRgnBox() const;

返回一个矩形区域。

bool operator ==(const TRegion& other) const;

运算符重载，如果两个区域相等返回 true。

bool operator !=(const TRegion& other) const;

运算符重载，如果两个区域不相等返回 true。

TRegion& operator =(const TRegion& source);

运算符重载，赋值运算。

TRegion& operator +=(const TSize& delta);

运算符重载，将另外的尺寸加到当前区域中并返回结果区域。

TRegion& operator -=(const TSize& delta);

运算符重载，将当前区域减去指定的大小。

TRegion& operator -=(const TRegion& source);

运算符重载，将当前区域减去指定的区域。

TRegion& operator &=(const TRegion& source);

运算符重载，将当前区域与所给区域相“与”。

TRegion& operator &=(const TRect& source);

运算符重载，将当前区域与所给矩形相“与”。

TRegion& operator |=(const TRegion& source);

运算符重载，将当前区域与所给区域相“或”。

TRegion& operator |=(const TRect& source);

运算符重载，将当前区域与所给矩形相“或”。

TRegion& operator ^=(const TRegion& source);

运算符重载，将当前区域与所给区域相“异或”。

TRegion& operator ^=(const TRect& source);

运算符重载，将当前区域与所给矩形相“异或”。

operator HRGN() const;

类型转换操作。

void SetRectRgn(const TRect& rect);

建立一个所给大小的矩形。

bool Touches(const TRect& rect) const;

如果当前区域碰到了所给的区域就返回 true。

A.50　TBitmap 类

头文件：gdiobjec.h

描述：TBitmap 类是 TGdiObject 类的派生类，它可以根据多种数据类型创建 GDI 位图对象。TBitmap 类是与设备有关的位图 DDB，不同于与设备无关的位图 DIB(TDib 对象)。TBitmat 类在 OWL 5.0 中的位置如图A-42所示。

Public 构造函数：

TBitmap(HBITMAP handle, TAutoDelete autoDelete = NoAutoDelete);

根据所给的位图句柄创建有关位图对象，设置 Handle 的值为所给的位图句柄的值；设置 ShouldDelete

的值为所给的 autoDelete 的值，缺省时为 false。注意：在 C++对象删除的时候位图句柄没有被删除。

TBitmap(const TClipboard& clipboard);

从剪贴板中获得数据创建位图。

TBitmap(const TBitmap& bitmap);

创建有关所给位图的拷贝。

TBitmap(int width, int height, uint8 planes=1, uint8 bitCount=1, void far * bits=0);

根据 bits 中的提供的位信息创建位图，高度、宽度和其它参数都按照所给的值设置。

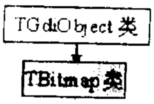


图 A-42 TBitmap 类在 OWL 5.0 中的位置

TBitmap(const BITMAP far * bitmap);

根据所给的位图结构的数据创建位图对象。

TBitmap(const TDC& Dc, int width, int height, bool discardable = false);

为所给的设备上下文创建位图。

TBitmap(const TDC& Dc, const TDib& dib, uint32 usage=CBM_INIT);

为所给的设备上下文根据所给的设备无关位图创建位图对象。

TBitmap(const TMetaFilePict& metaFile, TPalette& palette, const TSize& size);

根据所给的 metaFile 和调色板以及大小创建位图对象。

TBitmap(const TDib& dib, const TPalette * palette = 0);

根据所给的与设备无关的位图、调色板创建位图对象。

TBitmap(HINSTANCE instance, TResID resID);

根据所给的资源为指定的应用程序创建位图。

Public 成员函数：

uint8 BitsPixel() const;

返回位图中每一个像素点所占的位数。

uint32 GetBitmapBits(uint32 count, void far * bits) const;

将 count 指定大小的位图中的位拷贝到 bits 指定的缓冲区中。

bool GetBitmapDimension(TSize& size) const;

获得位图的大小。如果成功就返回 true，否则返回 false。

bool GetObject(BITMAP far& bitmap) const;

获得位图的信息(高度、宽度和颜色格式)，并将位图拷贝到 BITMAP 结构中。

int Height() const;

获得位图的高度信息。

TClipboard& operator <<(TClipboard& clipboard, TBitmap& bitmap);

将指定的位图拷贝到指定的剪贴板中。

operator HBITMAP() const;

类型转换操作。就位图句柄转换为 HBITMAP 结构。

uint8 Planes() const;

返回位图中的 plane 数目。

uint32 SetBitmapBits(uint32 count, const void far * bits);

将 bits 中的位图拷贝到当前位图中。

bool SetBitmapDimension(const TSize& size, TSize far * oldSize=0);

设置位图的大小并将原来的大小信息保存在 oldSize 参数中。如果成功返回 true，否则返回 false。

void ToClipboard(TClipboard& clipboard);

将位图拷贝到指定的剪贴板中。

int Width() const;

返回位图的宽度。

Protected 构造函数:

TBitMap();

Protected 成员函数:

void Create(const TDib& dib, const TPalette &palette);

根据所给的信息创建位图。

void Create(const TBitmap& src);

根据所给的信息创建位图。

A.51　TFont 类

头文件: gdiobjec.h

描述: TFont 类是 TGdiObject 的派生类,可以根据提供的信息创建字体对象。TFont 类在 OWL 5.0 中的位置如图 A-43 所示。

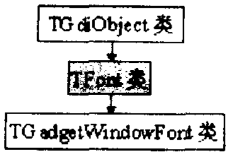


图 A-43　TFont 类在 OWL 5.0 中的位置

Public 构造函数:

TFont(HFONT handle, TAutoDelete autoDelete = NoAutoDelete);

用所给的字体句柄创建字体对象,设置 Handle 为所给的字体句柄,设置 ShouldDelete 为 false。注意:在 C++对象删除的时候字体句柄没有被删除。

```
TFont(const char far * facename = 0,
        int height = 0,
        int width = 0,
        int escapement = 0,
        int orientation = 0,
        int weight = FW_NORMAL,
        uint8 pitchAndFamily = DEFAULT_PITCH|FF_DONTCARE,
        uint8 italic = false,
        uint8 underline = false,
        uint8 strikeout = false,
        uint8 charSet = 1,
        uint8 outputPrecision = OUT_DEFAULT_PRECIS,
        uint8 clipPrecision = CLIP_DEFAULT_PRECIS,
        uint8 quality = DEFAULT_QUALITY);
```

根据所给的值创建字体对象。

```
TFont(int height, int width,
        int escapement = 0,
```

```
int orientation = 0,
int weight = FW_NORMAL,
uint8 italic = false,
uint8 underline = false,
uint8 strikeout = false,
uint8 charSet = 1,
uint8 outputPrecision = OUT_DEFAULT_PRECIS,
uint8 clipPrecision = CLIP_DEFAULT_PRECIS,
uint8 quality = DEFAULT_QUALITY,
uint8 pitchAndFamily = DEFAULT_PITCH|FF_DONTCARE,
const char far * facename = 0);
```

根据所给的值创建字体对象。构造函数中包括的缺省值与 Windows API 中的缺省值一致。

TFont(const LOGFONT far * logFont);

根据所给的 logFont 参数创建字体对象。

TFont(const TFont& font);

创建一个字体对象的拷贝。

Public 成员函数：

bool GetObject(LOGFONT far& logFont) const;

获得当前字体对象的信息并将它放在所给的 LOGFONT 结构中。如果成功就返回 true，否则返回 false。

operator HFONT() const;

类型转换操作，将字体句柄转换为 HFONT 类型。

A.52　TGadgetWindowFont 类

头文件：gadgetwi.h

描述：TGadgetWindowFont 类是 TFont 的派生类，提供了小器件窗口所使用的字体。TGodgetWindowFont 类可用于定义字体的大小和风格，它在 OWL 5.0 中的位置如图 A－44 所示。表 A－49 所列的 FF_xxxx 格式的值描述了字体的种类，表 A－50 则描述了字体的粗细。

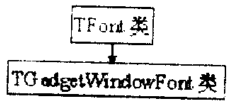


图 A－44　TGadgetWindowsFont 类在 OWL 5.0 中的位置

表 A－49

取　值	意　义
FF_DECORATIVE	比如 Old English 这样的特定字体
FF_DONTCARE	不管是什么字体都没关系
FF_MODERN	比较现代的字体
FF_ROMAN	像 Times New Roman 之类的字体
FF_SCRIPT	像 Script 之类的手写体字体
FF_SWISS	像 MS Sans Serif 之类的字体

表 A-50

取　值	意　义
FW_DONTCARE	字体的粗细没有关系
FW_THIN	细一些
FW_EXTRALIGHT	特别淡
FW_ULTRALIGHT	特别淡
FW_LIGHT	淡一些
FW_NORMAL	普通
FW_REGULAR	普通字体粗细
FW_MEDIUM	中等
FW_SEMIBOLD	稍微粗一些
FW_DEMIBOLD	稍微粗一些
FW_BOLD	粗一些
FW_EXTRBOLD	特别粗
FW_ULTRBOLD	特别粗
FW_BLACK	非常黑
FW_HEAVY	非常黑

Public 构造函数：

TGadgetWindowFont(int pointSize = 10, bool bold = false, bool italic = false);

构造一个 TGadgetWindowFont 对象。

A.53　TPalette 类

头文件：gdiobjec.h

描述：TPalette 类是 TGdiObject 类的派生类，它构造了 GDI 调色板对象。TPalbtte 类在 OWL 5.0 中的位置如图 A-45 所示。

Public 构造函数：

TPalette (HPALETTE handle, TAutoDelete autoDelete = NoAutoDelete);

构造一个调色板对象，设置 Handle 的值为所给的调色板句柄，设置 ShouldDelete 的值为 false。注意：在 C++对象删除的时候调色板句柄没有被删除。

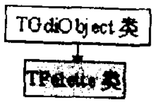


图 A-45　TPalette 类在 OWL 5.0 中的位置

TPalette(const TClipboard&);

从剪贴板中获得数据创建调色板对象。

TPalette(const TPalette& palette);

创建一个剪贴板的拷贝。

TPalette(const LOGPALETTE far * logPalette);

从所给的 logPalette 数组构造一个调色板对象。

TPalette(const PALETTEENTRY far * entries, int count);

根据所给的入口 entries 创建调色板对象。

TPalette(const BITMAPINFO far * info, uint flags = 0);

根据 BITMAPINFO 结构中的颜色表创建一个调色板对象。

TPalette(const BITMAPCOREINFO far * core, uint flags = 0);

根据 BITMAPCOREINFO 结构中的颜色表创建一个调色板对象。

TPalette(const TDib& dib, uint flags = 0);

根据所给的与设备无关的位图 dib 创建一个调色板对象。

TPalette();

只用于 Presentation 管理器，使用缺省的系统调色板创建一个调色板对象。

Public 成员函数：

void AnimatePalette(uint start, uint count, const PALETTEENTRY far * entries);

根据 PALETTEENTRY 结构数组的信息替换逻辑调色板的入口。

uint GetNearestPaletteIndex(TColor color) const;

获得与所给颜色最相近的调色板中颜色的索引。

uint GetNumEntries() const;

返回调色板的入口数目。

bool GetObject(uint16 far& numEntries) const;

找到逻辑调色板的入口数目并设置 numEntries 参数的值。如果成功就返回 true，否则返回 false。

uint GetPaletteEntries(uint16 start, uint16 count, PALETTEENTRY far * entries) const;

获得逻辑调色板的入口范围并将其赋值给 entries 数组。

uint GetPaletteEntry(uint16 index, PALETTEENTRY far& entry) const;

获得所给索引的逻辑调色板的入口并将它置于 entries 数组中。

TClipboard& operator << (TClipboard& clipboard, TPalette& palette);

将所给的调色板拷贝到指定的剪贴板中。

operator HPALETTE() const;

类型转换操作。就调色板句柄转换为 HPALETTE 类型。

bool ResizePalette(uint numEntries);

将逻辑调色板的大小重新定义为所给的值 numEntries。如果成功就返回 true，否则返回 false。

uint SetPaletteEntries(uint16 start, uint16 count, const PALETTEENTRY far * entries);

数组调色板中的 RGB 颜色值设置为 PALETTEENTRY 结构的 entries 数组中的值。

uint SetPaletteEntry(uint16 index, const PALETTEENTRY far& entry);

根据 entry 参数设置 index 指定的调色板入口的 RGB 值。

void ToClipboard(TClipboard& clipboard);

将此调色板拷贝到指定的剪贴板中。

bool UnrealizeObject();

在下一次调用 RealizePalette(HDC) 或 TDC::RealizePalette 函数时将逻辑调色板指向系统调色板。

Protected 成员函数：

void Create(const BITMAPINFO far * info, uint flags);

根据所给的位图信息设置调色板的值。

void Create(const BITMAPCOREINFO far * core, uint flags);

根据所给的位图信息设置调色板的值。

A.54 TBrush 类

头文件: gdiobjec.h

描述: TBrush 类是 TGdiObject 类的派生类，可以根据所给信息创建不同风格的刷子对象。TBrush 类在 OWL 5.0 中的位置如图 A-46 所示。

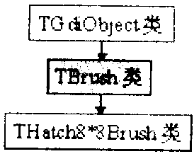


图 A-46 TBrush 类在 OWL 5.0 中的位置

Public 构造函数:

TBrush (HBRUSH handle, TAutoDelete autoDelete = NoAutoDelete);

创建一个刷子对象，设置 Handle 的值为所给的刷子句柄，设置 ShouldDelete 的缺省值为 false。注意：在 C++对象删除的时候刷子句柄没有被删除。

TBrush(TColor color);

使用所给的颜色创建一个具有 solid 风格的刷子。

TBrush(TColor color, int style);

按照所给的颜色和风格创建一个刷子。

TBrush(const TBitmap& pattern);

使用所给的调色板创建一个刷子。

TBrush(const TDib& pattern);

根据所给的与设备无关的位图创建刷子。

TBrush(const LOGBRUSH far * logBrush);

根据 logBrush 中的值创建刷子。

Public 成员函数:

bool GetObject(LOGBRUSH far& logBrush) const;

获得刷子对象的信息并放在所给的 LOGBRUSH 结构中。如果成功就返回 true，否则返回 false。

operator HBRUSH const();

类型转换操作。将刷子句柄转换为 HBRUSH 类型。

A.55 TPen 类

头文件: gdiobjec.h

描述: TPen 类是 TGdiObject 类的派生类，可创建 GDI 画笔，它在 OWL 5.0 中的位置如图A-47所示。

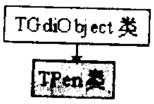


图 A-47 TPen 类在 OWL 5.0 中的位置

Public 构造函数:

TPen (HPEN handle, TAutoDelete autoDelete = NoAutoDelete);

创建一个画笔对象，设置 Handle 的值为所给的画笔句柄，设置 ShouldDelete 的缺省值为 false。注意：在 C++对象删除的时候画笔句柄没有被删除。

TPen(TColor color, int width=1, int style=PS_SOLID);

根据所给的值创建一个画笔对象。style 的值和它所代表的含义如表 A-51 所示。

表 A-51

取　值	意　　义
PS_SOLID	创建一个 solid 风格的画笔
PS_DASH	创建一个 dashed 风格的画笔
PS_DOT	创建一个 dotted 风格的画笔
PS_DASHDOT	创建一个点划线风格的画笔
PS_DASHDOTDOT	创建一个双点划线风格的画笔
PS_NULL	创建一个空画笔
PS_INSIDEFRAME	创建一个 solid 风格的画笔。它可以自动适应一个区域的大小

TPen(const LOGPEN far * logPen);

根据 logPen 的值创建一个画笔对象。

TPen(const TPen& pen);

创建一个画笔对象的拷贝。

```
TPen(uint32 penStyle,
     uint32 width,
     const TBrush& brush,
     uint32 styleCount,
     uint32 style);
```

(32 位应用程序中)根据所给的值创建一个画笔对象。

```
TPen(uint32 penStyle,
     uint32 width,
     const LOGBRUSH& logBrush,
     uint32 styleCount,
     uint32 style);
```

(32 位应用程序中)根据所给的值创建一个画笔对象。

Public 成员函数：

bool GetObject(LOGPEN far& logPen) const;

获得画笔对象的信息并置于所给的 LOGPEN 结构中。如果成功就返回 true，否则返回 false。

operator HPEN() const;

类型转换操作，将一个画笔句柄转换为 HPEN 类型。

A.56　TIcon 类

头文件：gdiobjec.h

描述：TIcon 类是 TGdiObject 类的派生类，可以创建 GDI 图标对象。因为图标本身并不是 GDI 真正的对象，所以 TIcon 类包含了一个析构函数 ~TGdiObject。TIcon 类在 OWL 5.0 中的位置如图 A-48 所示。

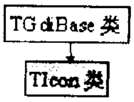


图 A-48　TIcon 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

TIcon(HICON handle, TAutoDelete autoDelete = NoAutoDelete);

创建一个图标对象，设置 Handle 的值为所给的图标句柄，设置 ShouldDelete 的缺省值为 false。注意：在 C++对象删除的时候图标句柄没有被删除。

TIcon(HINSTANCE instance, const TIcon& icon);

创建一个图标对象的拷贝。

TIcon(HINSTANCE instance, TResID resID);

根据所给的资源创建一个图标。

TIcon(HINSTANCE instance, const char far * filename, int index);

从所给的资源文件中创建一个图标对象。

TIcon(HINSTANCE instance, const TSize& size, int planes, int bitsPixel, const void far * andBits, const void far * xorBits);

用所给的值创建一个图标对象。

TIcon(const void * resBits, uint32 resSize);

根据 resBits 位图和所给的大小创建一个图标对象。

TIcon(const ICONINFO * iconInfo);

根据所给的 ICONINFO 类型的值创建一个图标对象。

~TIcon();

删除图标并释放图标占用的内存。

Public 成员函数：

bool GetIconInfo(ICONINFO * iconInfo) const;

获得图标对象的信息并置于 iconInfo 中，如果成功返回 true，否则就返回 false。

operator HICON() const;

类型转换操作。将图标句柄转换为 HICON 类型。

A.57　TCursor 类

头文件：gdiobjec.h

描述：TCursor 类是 TGdiObject 类的派生类，可以创建光标对象。因为光标本身并不是 GDI 真正的对象，所以 TCursor 类包含了一个析构函数 ~ TGdiBase。TCursor 类在 OWL 5.0 中的位置如图 A-49 所示。

Public 构造函数和析构函数：

TCursor(HCURSOR handle, TAutoDelete autoDelete = NoAutoDelete);

创建一个光标对象，设置 Handle 的值为所给的光标句柄，设置 ShouldDelete 的缺省值为 false。注意：在 C++ 对象删除的时候光标句柄没有被删除。

TCursor(HINSTANCE instance, const TCursor& cursor);

创建一个光标对象的拷贝。

TCursor(HINSTANCE instance, ResId resId);

根据所给的资源 ID 值创建光标对象。

```
TCursor(HINSTANCE instance,
        const TPoint& hotSpot,
        const TSize& size,
        void far * andBits,
        void far * xorBits);
```

在指定的一点创建一个指定大小的光标对象。

TCursor(const void * resBits uint32 resSize);

(用于 32 位应用程序)根据指定资源创建光标对象。

TCursor(const ICONINFO * iconInfo);

(用于 32 位应用程序)根据 ICONINFO 结构中的值创建光标对象。

~TCursor();

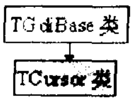


图 A-49　TCursor 类在 OWL 5.0 中的位置

Public 成员函数：

bool *GetIconInfo(ICONINFO* * iconInfo) const;

获得光标对象的信息并置于 iconInfo 中。

operator HCURSOR() const;

类型转换操作，将光标句柄转换为 HCURSOR 类型。

bool operator ==(const TCursor& other) const;

如果两个光标对象相等就返回 true，否则返回 false。

A.58　TDib 类

头文件： gdiobjec.h

描述： TDib 类是 TGdiObject 类的派生类，可以创建与设备无关的位图 DIB(Device Independent Bitmap)对象。与设备无关的位图没有 Windows 句柄，但包含了位图的结构信息、调色板和位图中的位信息，使得使用与设备无关位图和使用其他 GDI 对象一样方便。因为与设备无关位图本身并不是真正的 GDI 对象，所以 TDib 类包含了一个析构函数。TDib 类在 OWL 5.0 中的位置如图 A-50 所示。

类型定义：

enum Map {MapFace, MapText, MapShadow, MapHighlight, MapFrame};

枚举了窗口中要设置颜色部分的值。

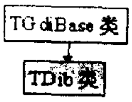


图 A-50　TDib 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

TDib(HGLOBAL handle, TAutoDelete autoDelete = NoAutoDelete);

创建一个 DIB 对象，设置 Handle 的值为所给的 DIB 句柄，设置 ShouldDelete 的缺省值为 false。注意：在

C++对象删除的时候 DIB 句柄没有被删除。

TDib(const TClipboard& clipboard);

使用所给的剪贴板创建 DIB 对象。

TDib(const TDib& dib);

创建所给 DIB 对象的拷贝。

TDib(int width, int height, int nColors, uint16 mode = DIB_RGB_COLORS);

使用所给的参数创建一个 DIB 对象。

TDib(HINSTANCE instance, TResID resID);

使用所给的资源创建一个 DIB 对象。

TDib(const char * name);

使用所给的资源文件创建一个 DIB 对象。

TDib(const TBitmap& bitmap, const TPalette * pal = 0);

使用所给的位图和调色板创建一个 DIB 对象。

~TDib();

Public 成员函数：

operator const BITMAPINFO far * () const;

将 DIB 转换为位图信息结构 BITMAPINFO，其中包含位图的颜色格式、尺寸等信息。

operator BITMAPINFO far * ();

将 DIB 转换为位图信息结构 BITMAPINFO，其中包含位图的颜色格式、尺寸等信息。

operator const BITMAPINFOHEADER far * () const;

将 DIB 转换为位图信息。

operator BITMAPINFOHEADER far * ();

将 DIB 转换为位图信息。

bool ChangeModeToPal(const TPalette& pal);

将当前的颜色表转换为调色板中相应的值，如果成功就返回 true，否则就返回 false。

bool ChangeModeToRGB(const TPalette& pal);

将当前的颜色表转换为 RGB 值。如果成功就返回 true，否则就返回 false。

int FindColor(TColor color);

返回所给的颜色在调色板中的入口。

int FindIndex(uint16 index);

返回所给索引在调色板中的入口。

const void HUGE * GetBits() const;

获得 DIB 位图的全部位信息。

void HUGE * GetBits();

获得 DIB 位图的全部位信息。

TColor GetColor(int entry) const;

返回所给调色板入口对应的颜色

const TRgbQuad far * GetColors() const;

获得 DIB 位图中位的颜色信息。

TRgbQuad far * GetColors();

获得 DIB 位图中位的颜色信息。

uint16 GetIndex(int entry) const;

返回所给的调色板入口对应的索引。

const uint16 far * GetIndices() const;

获得 DIB 位图中全部位颜色的索引。

uint16 far * GetIndices();

获得 DIB 位图中全部位颜色的索引。

const BITMAPINFO far * GetInfo() const;

返回 DIB 位图的信息。BITMAPINFO 结构包含了位图的尺寸和颜色信息。

BITMAPINFO far * GetInfo();

返回 DIB 位图的信息。BITMAPINFO 结构包含了位图的尺寸和颜色信息。

const BITMAPINFOHEADER far * GetInfoHeader() const;

返回 DIB 位图的信息头。BITMAPINFO 结构包含了 DIB 位图的颜色和尺寸。

BITMAPINFOHEADER far * GetInfoHeader();

返回 DIB 位图的信息头。BITMAPINFO 结构包含了 DIB 位图的颜色和尺寸。

operator HANDLE() const;

将 DIB 类型转换为 HANDLE 类型。

int Height() const;

返回 DIB 位图的高度。

bool IsOK() const;

如果 info 的值为 0 就返回 false，否则返回 true。

bool IsPM() const;

如果 IsCore 的值为 true 就返回 true，否则返回 false。

int MapColor(TColor fromColor, TColor toColor, bool doAll = false);

将当前 DIB 位图中调色板中的 toColor 设置为 fromColor。

int MapIndex(uint16 fromIndex, Word toIndex, bool doAll = false);

将当前 DIB 位图中调色板中的 toIndex 设置为 fromIndex。

int MapUIColors(uint mapColors, TColor * = 0);

将 UI 颜色设置为所给 mapColors 的值。

long NumColors() const;

返回 NumClrs，它是 DIB 调色板中的颜色数目。

uint16 NumScans() const;

返回 DIB 的扫描数目。

TClipboard& operator <<(TClipboard& clipboard, TDib& dib);

将 DIB 位图拷贝到所给的剪贴板中。

bool operator == (const TDib& other) const

如果两个 DIB 位图的句柄相同就返回 true。

void SetColor(int entry, TColor color);

为所给的调色板入口设置颜色。

void SetIndex(int entry, uint16 index);

为所给的调色板入口设置索引。

TSize Size() const;

返回 DIB 位图的大小。

uint16 StartScan() const;

返回 DIB 位图的起始扫描线。

void ToClipboard(TClipboard& clipboard);

将 DIB 位图输送到所给的剪贴板。

operator const TRgbQuad far * () const;

将 DIB 位图类型转换为一个颜色类型的指针。

operator TRgbQuad far * () const;

将 DIB 位图类型转换为一个颜色类型的指针。

uint16 Usage() const;

返回 DIB 位图的 Mode 的值。

int Width() const;

返回 DIB 位图的宽度。

bool WriteFile(const char * filename);

写入文件，如果成功就返回 true，否则就返回 false。

Protected 数据成员：

void HUGE * Bits;

Bits 指向位图的内存块。

int H;

位图以像素点为单位的高度。

BITMAPINFO far * Info;

位图信息。

bool IsCore;

如果 DIB 位图是一个过时的 PM DIB 就设置为 true，否则就设置为 false。

bool IsResHandle;

如果位图使用资源就设置为 true，否则就设置为 false。

uint16 Mode;

保存位图模式。

long NumClrs;

位图中的颜色数目。

int W;

位图以像素点为单位的宽度。

Protected 成员函数：

void InfoFromHandle();

锁定 DIB 位图的句柄，获得位图信息。

bool LoadFile(const char * name);

从所给的文件获得位图。

bool LoadResource(HINSTANCE instance, TResID resID);

从所给的资源获得位图。

bool Read(TFile& file, long offBits = 0);

从文件中 offBits 个字节以后开始的读出位图。如果成功就返回 true，否则就返回 false。

A.59　TDC 类

头文件：dc.h

描述：TDC 是 GDI 设备上下文类的基类，它包括了设备上下文的基本特征。由于 TDC 类是从 TGdiBase 类派生出来的，继承了一个设备上下文句柄 HDC 并包括了一个类型转换操作用来获得这个句柄，因此 TDC 的成员函数可以调用 Windows API 中有关 HDC 的函数。由 TDC 派生出来的子类包括：TWindowDC、TMemoryDC、TMetaFileDC、TDibDC 和 TPrintDC 等等，它们可以构造不同用途的设备上下文。

TDC 类的 protected 数据成员中包括了四个句柄：OrgBrush、OrgPen、OrgFont 和 OrgPalette，其中分别保存了设备上下文所选择的 GDI 对象。调用函数 SelectObject 和 SelectPalette 来选择新的对象，调用函数 RestoreBrush 和 RestorePen 来恢复过去的对象，当有关 TDC 对象被删除时，其中的所有对象都将被删除（如果 TDC:: ShouldDelete 的值为 false 时将不删除对象）。以上这些足以使你有充足的理由放弃原来的设备上下文的繁杂操作而使用 OWL 给你提供的 TDC 类。TDC 类在 OWL 5.0 中的位置如图 A－51 所示。

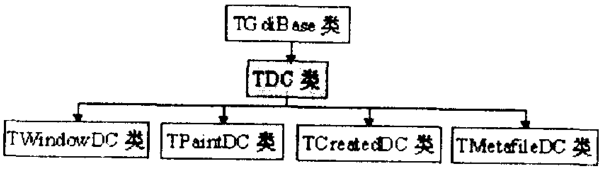


图 A－51　TDC 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

TDC(HDC handle);

virtual ~TDC();

Public 成员函数：

bool AngleArc(int x, int y, uint32 radius, float startAngle, float sweepAngle);

（只用于 32 位应用程序）根据所给的值绘制一条弧线及其弦，如果成功返回 true，否则返回false。

bool AngleArc(const TPoint& center, uint32 radius, float startAngle, float sweepAngle);

（只用于 32 位应用程序）根据所给的值绘制一条弧线及其弦，如果成功返回 true，否则返回false。

bool Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

根据所给的值绘制一条椭圆弧线，如果成功就返回 true，否则就返回 false。

bool Arc(const TRect& r, const TPoint& start, const TPoint& end);

根据所给的值绘制一条椭圆弧线，如果成功就返回 true，否则就返回 false。

bool BeginPath();

（只用于 32 位应用程序）打开一个新的设备上下文路径并删除旧的设备上下文。调用此函数后，应用程序就可以开始在设备上下文中绘图了。TDC 的成员函数中可以提供这个路径绘图的有：AngleArc、Arc、Chord、CloseFigure、Ellipse、ExtTextOut、LineTo、MoveToEx、Pie、PolyBezier、PolyBezierTo、PolyDraw、Polygon、Polyline、PolylineTo、PolyPolygon、PolyPolyline、Rectangle、RoundRect 和 TextOut 等。调用函数 TDC::EndPath 可以关闭路径，如果成功返回 true，否则就返回 false。

bool BitBlt(int dstX, int dstY, int w, int h,
　　const TDC& srcDC,
　　int srcX, int srcY,

uint32 rop=SRCCOPY);

将位块从所给的 srcDc 传输到这个 DC 中。

bool BitBlt(const TRect& dst,

const TDC& srcDC,

const TPoint& src,

uint32 rop=SRCCOPY);

将位块从所给的 srcDc 传输到这个 DC 中。

bool Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

用当前所选择的画笔和刷子在设备上下文中绘制填充一个弦。

bool Chord(const TRect& r, const TPoint& start, const TPoint& end);

用当前所选择的画笔和刷子在设备上下文中绘制填充一个弦。

bool CloseFigure();

(只用于 32 位应用程序)关闭一个 DC 中打开的图。

bool DPtoLP(TPoint * points, int count = 1) const;

将 points 数组中的设备点转换为逻辑点，如果成功就返回 true，否则就返回 false。

bool DrawFocusRect(int x1, int x2, int y1, int y2);

按照所给的参数在 DC 中绘制矩形，如果成功就返回 true，否则就返回 false。

bool DrawFocusRect(const TRect& rect);

按照所给的参数在 DC 中绘制矩形，如果成功就返回 true，否则就返回 false。

bool DrawIcon(int x, int y, const TIcon& icon);

在 DC 中绘制所给的图标，如果成功就返回 true，否则就返回 false。

bool DrawIcon(const TPoint& point, const TIcon& icon);

在 DC 中绘制所给的图标，如果成功就返回 true，否则就返回 false。

virtual bool DrawText(const char far * string, int count, const TRect& r, uint16 format = 0);

按照所给的格式和区域绘制所给的字符串，如果 count 的值是 -1 就绘制全部的字符。format 的取值和意义如表 A-52 所示。

表 A-52

取　值	意　　义
DT_BOTTOM	定义文本下边对齐。与 DT_SINGLELINE 一起使用
DT_CALCRECT	确定矩形的宽度和高度
DT_CENTER	文本居中对齐
DT_EXPANDTABS	控件的 tab 键码。缺省时，一个 tab 键是 8 个字符的宽度
DT_EXTERNALLEADING	在行高中包括字体额外的高度
DT_LEFT	文本居左对齐
DT_NOCLIP	不剪贴绘制文本
DT_NOPREFIX	关闭字体前缀
DT_RIGHT	文本居右对齐
DT_SINGLELINE	定义单行文本
DT_TABSTOP	设置 tab 站
DT_TOP	文本上边对齐
DT_VCENTER	文本纵向居中对齐
DT_WORDBREAK	定义断词

注意：DT_CALCRECT、DT_EXTERNALLEADING、DT_INTERNAL、DT_NOCLIP 和 DT_NOPREFIX 不能和 DT_TABSTOP 一起使用。DrawText 函数使用 DC 当前选择的字体、文本颜色和背景颜色绘制文本。如果不使用 DT_NOCLIP 格式，DrawText 在显示文本时就会切割文本以保证不会超出矩形区域；如果不使用 DT_SINGLELINE，文本就被认为是多行文本。函数执行成功就返回 true，否则就返回 false。

bool Ellipse(int x1, int y1, int x2, int y2);

在 DC 中使用当前选择的画笔和刷子对象绘制并填充一个椭圆，如果成功返回 true，否则返回false。

bool Ellipse(const TPoint& p1, const TPoint& p2);

在 DC 中使用当前选择的画笔和刷子对象绘制并填充一个椭圆，如果成功返回 true，否则返回false。

bool Ellipse(const TPoint& point, const TSize& size);

在 DC 中使用当前选择的画笔和刷子对象绘制并填充一个椭圆，如果成功返回 true，否则返回false。

bool Ellipse(const TRect& rect);

在 DC 中使用当前选择的画笔和刷子对象绘制并填充一个椭圆，如果成功返回 true，否则返回false。

bool EndPath();

（只用于 32 位应用程序）关闭 DC 中定义的绘图路径，如果成功就返回 true，否则就返回 false。

```
int EnumFontFamilies(const char far * family,
                     FONTENUMPROC proc,
                     void far * data) const;
```

枚举 family 中定义且在这个 DC 中可以用的字体。

```
int EnumFonts(const char far * faceName,
              OLDFONTENUMPROC callback,
              void far * data) const;
```

根据所给的 faceName 的值枚举 DC 中可用的字体。

```
int EnumMetaFile(const TMetaFilePict& metaFile,
                 MFENUMPROC callback,
                 void * data) const;
```

枚举 GDI 中使用 metaFile 的调用。

```
int EnumObjects(uint objectType,
                GOBJENUMPROC proc,
                void far * data) const;
```

枚举这个 DC 中可以使用的画笔和刷子对象。

int ExcludeClipRect(const TRect& rect);

为这个 DC 创建一个新的剪贴区域。新的区域包含了除去所给区域 rect 后的当前的剪贴区域。返回值表明了当前的剪贴区域的类型(见表 A-53)。

表 A-53

区　　域	意　　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

int ExcludeUpdateRgn(HWND wnd);

提供排除 DC 剪贴区域中的更新区域来避免绘制窗口中的失效区域。返回值表明了当前剪贴区域的类型，见表 A-54 所示。

表 A-54

区　域	意　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

bool ExtFloodFill(const TPoint& point, TColor color, uint16 fillType);

在 DC 中填充一个区域。填充从所给的点 point 开始，使用当前选择的刷子对象，color 表明了填充区域的边框颜色。fillType 参数决定了填充方式，见表 A-55 所示。如果填充成功返回 true，否则返回false。

表 A-55

取　值	意　义
FLOODFILLBORDER	填充区域的边框绘制为所给的颜色
FLOODFILLSURFACE	填充区域的边框用所给的颜色决定，填充将从一点开始直到碰到了所给的颜色

```
virtual bool ExtTextOut(int x, int y,
            uint16 options,
            const TRect * r,
            const char far * string,
            int count,
            const int far * dx = 0);
```

按照所给的方式和区域绘制所给的字符串，如果 count 的值是-1 就绘制全部的字符。

```
bool ExtTextOut(const TPoint& p,
            uint16 options,
            const TRect * r,
            const char far * string,
            int count,
            const int far * dx = 0);
```

按照所给的方式和区域绘制所给的字符串，如果 count 的值是-1 就绘制全部的字符。

bool FillPath();

(只用于 32 位应用程序)关闭 DC 中当前路径中的图并使用当前的设置和填充模式来填充路径。然后删除一个路径。如果成功就返回 true，否则就返回 false。

bool FillRect(int x1, int y1, int x2, int y2, const TBrush& brush);

用所给的刷子填充这个 DC 中所给的区域，如果成功就返回 true，否则就返回 false。

bool FillRect(const TRect& rect, const TBrush& brush);

用所给的刷子填充这个 DC 中所给的区域，如果成功就返回 true，否则就返回 false。

bool FillRgn(const TRegion& region, const TBrush& brush);

使用这个DC中所选择的刷子填充所给的区域，如果成功就返回true，否则就返回false。

bool FlattenPath();

(只用于32位应用程序)转换DC中当前选择路径中的曲线，如果成功返回true，否则返回false。

bool FloodFill(const TPoint& point, TColor color);

在DC中填充一个区域，填充从point指定点开始并使用当前选择的刷子，如果填充成功返回true，否则返回false。color参数定义了区域边框的颜色。

bool FrameRect(int x1, int x2, int y1, int y2, const TBrush& brush);

在DC中给所给的矩形rect绘制边框，如果成功返回true，否则就返回false。

bool FrameRect(const TRect& rect, const TBrush& brush);

在DC中给所给的矩形rect绘制边框，如果成功就返回true，否则就返回false。

bool FrameRgn(const TRegion& region, const TBrush& brush, const TPoint& p);

在DC中给所给的区域region绘制边框，如果成功就返回true，否则就返回false。

bool GetAspectRatioFilter(TSize& size) const;

获得DC中当前的横纵比。

TColor GetBkColor() const;

获得DC中当前的背景颜色。

int GetBkMode() const;

返回DC的背景模式。背景模式包括OPAQUE和TRANSPARENT。

bool GetBoundsRect(TRect& bounds, uint16 flags) const;

获得DC中当前边界的信息，如果成功返回true，否则就返回false。

bool GetBrushOrg(TPoint& point) const;

获得DC中当前刷子的初始位置，如果成功就返回true，否则就返回false。

bool GetCharABCWidths(uint firstChar, uint lastChar, ABC * abc);

获得获得用当前字体绘制的字符串中firstChar和lastChar之间每一个字符的宽度并存放在ABC结构的数组中，如果成功就返回true，否则就返回false。

bool GetCharWidth(uint firstChar, uint lastChar, int * buffer);

获得获得用当前字体绘制的字符串中firstChar和lastChar之间每一个字符的逻辑宽度并存放在所给的缓冲区中，如果成功就返回true，否则就返回false。

int GetClipBox(TRect& rect) const;

设置DC中当前的剪贴框的大小。返回值与意义如表A-56所示。

表A-56

区　域	意　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

TRect GetClipBox() const;

设置DC中当前的剪贴框的大小。返回值与意义如表A-57所示。

表 A－57

区　　域	意　　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

bool GetClipRgn(TRegion& region) const;

获得 DC 中当前的剪贴区域，如果成功就返回 true，否则就返回 false。

HANDLE GetCurrentObject(uint objectType) const;

返回 DC 中当前选择的对象句柄。objectType 说明了需要获得的对象种类，它的值可以是 OBJ_PEN、OBJ_BRUSH、OBJ_PAL、OBJ_FONT 或者 OBJ_BITMAP。如果调用失败就返回 0。

bool GetCurrentPosition(TPoint& point) const;

获得 DC 中当前点的坐标，如果成功就返回 true，否则就返回 false。

bool GetDCOrg(TPoint& point) const;

获得设备上下文的最终转换原点，如果成功就返回 true，否则就返回 false。

virtual int GetDeviceCaps(int index) const;

(用于 Windows 3.1 及以后的版本)返回 DC 的能力信息。index 参数给出了所需的信息种类。

```
bool GetDIBits(const TBitmap& bitmap,
               uint16 startScan,
               uint16 numScans,
               void HUGE * bits,
               const BITMAPINFO far& info,
               uint16 usage)
```

获得一些或全部的 DC 中所给位图的位并将它们拷贝到 info 定义的与设备无关位图 DIB 格式的 BITMAPINFO 类型中，如果成功就返回 true，否则就返回 false。usage 决定了 BITMAPINFO 结构中 bmiColors 参数的格式，它的取值和意义如表 A－58 所示。

表 A－58

取　　值	意　　义
DIB_PAL_COLORS	当前逻辑调色板中的颜色表是一个 16 位索引组成的数组
DIB_RGB_COLORS	颜色表包含了 RGB 的值
DIB_PAL_INDICES	位图中没有颜色表。DIB 位图的位是采用相同调色板的索引

bool GetDIBits(const TBitmap& bitmap, TDib& dib);

获得一些或全部的 DC 中 bitmap 所给位图的位，并将它们拷贝到 dib 定义的与设备无关位图 DIB 中。如果成功就返回 true，否则就返回 false。

uint32 GetFontData(uint32 table, uint32 offset, void * buffer, long data);

根据所给的字体文件获得字体属性并拷贝到所给的缓冲区 buffer 中，如果成功就返回 buffer 中的字节数，否则就返回－1。

```
uint32 GetGlyphOutline(uint chr,
            uint format,
            GLYPHMETRICS far& gm,
            uint32 buffSize,
            void far * buffer,
            const MAT2 far& mat2);
```

获得 DC 中所给字符 chr 的字体尺寸和其他数据。如果调用成功就返回一个正数，否则就返回 GDI_ERROR。format 参数的取值和意义如表 A-59 所示。

表 A-59

取　值	意　　义
1	获得字符笔画位图
2	获得光栅化了的局部格式的曲线数据(字体的设计单位)

```
int GetKerningPairs(int pairs, KERNINGPAIR far * krnPair);
```

获得 DC 中当前字体的 kerning pairs 并拷贝到 krnPair 所指的 KERNINGPAIR 结构中。如果成功就返回获得 pairs 的实际数目；否则就返回 0。

```
int GetMapMode() const;
```

获得当前的映射模式。如果成功就返回当前窗口 DC 的映射模式，否则就返回 0。mode 取值和意义如表 A-60 所示。

表 A-60

取　值	意　　义
MM_ANISOTROPIC	X 和 Y 为独立标度的任意单位
MM_HIENGLISH	每一个逻辑单位为 0.001 英寸。坐标原点在窗口的左下角
MM_HIMETRIC	每一个逻辑单位为 0.01 毫米。坐标原点在窗口的左下角
MM_ISOTROPIC	X 和 Y 为统一标度的任意单位
MM_LOENGLISH	每一个逻辑单位为 0.01 英寸。坐标原点在窗口的左下角
MM_LOMETRIC	每一个逻辑单位为 0.1 毫米。坐标原点在窗口的左下角
MM_TEXT	每一个逻辑单位一个物理像素点。坐标原点在窗口的左上角
MM_TWIPS	每一个逻辑单位为 1/1440 英寸。坐标原点在窗口的左下角

```
TColor GetNearestColor(TColor Color) const;
```

获得 DC 中的调色板中与所给的颜色最接近的颜色值。

```
uint32 GetOutlineTextMetrics(uint data, OUTLINETEXTMETRIC far& otm);
```

获得 DC 中 TrueType 字体的属性数据并拷贝到 otm 所指的 OUTLINETEXTMETRIC 数组中。如果成功返回 true，否则返回 false。OUTLINETEXTMETRIC 结构的定义如下：

```
typedef struct OUTLINETEXTMETRIC {
    UINT            otmSize;
    TEXTMETRIC      otmTextMetrics;
```

```
BYTE        otmFiller;
PANOSE      otmPanoseNumber;
UINT        otmfsSelection;
UINT        otmfsType;
int         otmsCharSlopeRise;
int         otmsCharSlopeRun;
int         otmItalicAngle;
UINT        otmEMSquare;
int         otmAscent;
int         otmDescent;
UINT        otmLineGap;
UINT        otmsCapEmHeight;
UINT        otmsXHeight;
RECT        otmrcFontBox;
int         otmMacAscent;
int         otmMacDescent;
UINT        otmMacLineGap;
UINT        otmusMinimumPPEM;
POINT       otmptSubscriptSize;
POINT       otmptSubscriptOffset;
POINT       otmptSuperscriptSize;
POINT       otmptSuperscriptOffset;
UINT        otmsStrikeoutSize;
int         otmsStrikeoutPosition;
int         otmsUnderscoreSize;
int         otmsUnderscorePosition;
PSTR        otmpFamilyName;
PSTR        otmpFaceName;
PSTR        otmpStyleName;
PSTR        otmpFullName;
} OUTLINETEXTMETRIC;
```

uint16 GetOutlineTextMetrics(uint data, OUTLINETEXTMETRIC far& otm);

获得 DC 中 TrueType 字体的属性数据并拷贝到 otm 所指的 OUTLINETEXTMETRIC 数组中。如果成功就返回 true，否则就返回 false。

TColor GetPixel(int x, int y) const;

获得所指像素点的颜色值。

TColor GetPixel(const TPoint& point) const;

获得所指像素点的颜色值。

int GetPolyFillMode() const;

返回 DC 中当前的多边形填充模式：ALTERNATE 或者 WINDING。

int GetROP2() const;

获得当前 DC 中的绘图模式。返回值的取值和意义如表 A-61 所示。

表 A-61

取　值	意　义
R2_BLACK	像素点总为黑色
R2_COPYPEN	像素点为画笔颜色
R2_MASKNOTPEN	像素点为显示颜色与反画笔颜色的与
R2_MASKPEN	像素点为显示颜色与画笔颜色的与
R2_MASKPENNOT	像素点为反显示颜色与画笔颜色的与
R2_MERGEPEN	像素点为显示颜色与画笔颜色的或
R2_MERGENOTPEN	像素点为显示颜色与反画笔颜色的或
R2_MERGEPENNOT	像素点为反显示颜色与画笔颜色的或
R2_NOP	像素点保持不变
R2_NOT	像素点是显示颜色的反
R2_NOTCOPYPEN	像素点是画笔颜色的反
R2_NOTMASKPEN	像素点是 R2_MASKPEN 值的反
R2_NOTMERGEPEN	像素点是 R2_MERGEPEN 值的反
R2_NOTXORPEN	像素点是 R2_XORPEN 值的反
R2_WHITE	像素点总是白色
R2_XORPEN	像素点是显示颜色和画笔颜色的异或

int GetStretchBltMode() const;

返回 DC 当前的缩放模式。返回值为 BLACKONWHITE、COLORONCOLOR 或者 WHITEONBLACK。

uint GetSystemPaletteEntries(int start, int num, PALETTEENTRY far * entries) const;

获得调色板中以 start 开始的入口，并拷贝到 entries 所指的 PALETTEENTRY 结构中。PALETTEENTRY 结构定义如下：

```
typedef struct tagPALETTEENTRY {
    BYTE peRed;     BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
    } PALETTEENTRY;
```

返回实际传输的调色板入口数目。

uint GetSystemPaletteUse() const;

确定一个应用程序是否已存取了全部的系统调色板。返回值为 SYSPAL_NOSTATIC 或者 SYSPAL_STATIC。

```
bool GetTabbedTextExtent(const char far * string,
            int stringLen,
            int numPositions,
            const int far * positions,
            TSize& size) const;
```

获得所给字符串输出时的高度和宽度，计算时使用当前的字体。

```
TSize GetTabbedTextExtent(const char far * string,
            int stringLen,
            int numPositions,
            const int far * positions) const;
```

获得所给字符串输出时的高度和宽度，计算时使用当前的字体。

```
uint GetTextAlign() const;
```

如果成功就返回 DC 中当前字体对齐模式，否则就返回 GDI_ERROR。文本对齐模式的取值和意义如表 A-62 所示。

表 A-62

取　值	意　　义
TA_BASELINE	限定的矩形内字体的基线与 x 轴对齐
TA_BOTTOM	限定矩形的底边与 x 轴对齐
TA_TOP	限定矩形的顶部与 x 轴对齐
TA_CENTER	y 轴与限定矩形的中心对齐
TA_LEFT	y 轴与限定矩形的左边对齐
TA_RIGHT	y 轴与限定矩形的右边对齐
TA_NOUPDATECP	不更新当前的位置
TA_UPDATECP	更新当前位置

```
int GetTextCharacterExtra() const;
```

如果成功就返回 DC 中当前字符间距，否则就返回 INVALID_WIDTH。

```
TColor GetTextColor() const;
```

获得 DC 中当前文本的颜色。

```
bool GetTextExtent(const char far * string, int stringLen, TSize& size);
```

计算输出所给的字符串所需的高度和宽度。

```
TSize GetTextExtent(const char far * string, int stringLen);
```

计算输出所给的字符串所需的高度和宽度。

```
int GetTextFace(int count, char far * facename) const;
```

获得 DC 中当前的字体名称并拷贝到 facename 中。如果成功，GetTextFace 返回实际拷贝的字符数目，否则就返回 0。

```
bool GetTextMetrics(TEXTMETRIC far& metrics) const;
```

获得 DC 中当前选择的字体的量度，并拷贝到 metrics 所指的 TEXTMETRIC 结构中。TEXTMETRIC 结构的定义如下：

16 位的结构定义：

```
typedef struct tagTEXTMETRIC {
  int tmHeight;
  int tmAscent;
  int tmDescent;
  int tmInternalLeading;
```

```
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
} TEXTMETRIC;
```

32 位的结构定义如下：

```
typedef struct tagTEXTMETRIC {
    LONG tmHeight;
    LONG tmAscent;
    LONG tmDescent;
    LONG tmInternalLeading;
    LONG tmExternalLeading;
    LONG tmAveCharWidth;
    LONG tmMaxCharWidth;
    LONG tmWeight;
    LONG tmOverhang;
    LONG tmDigitizedAspectX;
    LONG tmDigitizedAspectY;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
} TEXTMETRIC;
```

返回 DC 中当前选择的字体的量度。

```
bool GetViewportExt(TSize& extent) const;
```

获得 DC 中视口的范围。

TSize GetViewportExt() const;

获得 DC 中视口的范围。

bool GetViewportOrg(TPoint& point) const;

获得 DC 中坐标的原点。

TPoint GetViewportOrg() const;

获得 DC 中坐标的原点。

bool GetWindowExt(TSize& extent) const;

获得与 DC 相关的窗口的大小。

TSize GetWindowExt() const;

获得与 DC 相关的窗口的大小。

bool GetWindowOrg(TPoint& point) const;

获得与 DC 相关的窗口的原点坐标。

TPoint GetWindowOrg() const;

获得与 DC 相关的窗口的原点坐标。

```
virtual bool GrayString(const TBrush& brush,
                GRAYSTRINGPROC outputFunc,
                const char far * string,
                int count,
                const TRect& r);
```

在指定的区域输出灰色文本。outputFunc 函数是应用程序通过的用来输出灰色文本的过程函数。

operator HDC() const{return Handle;}

类型转换操作。获得 DC 的 HDC 句柄。

int IntersectClipRect(const TRect& rect);

把 DC 中当前的剪贴区域与所给的区域的相交区域作为当前的剪贴区域。返回值为剪贴区域的类型，其意义见表 A-63。

表 A-63

区　域	意　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

bool InvertRect(int x1, int x2, int y1, int y2);

将指定的矩形区域颜色取反。如果成功就返回 true，否则就返回 false。

bool InvertRect(const TRect& rect);

将指定的矩形区域颜色取反。如果成功就返回 true，否则就返回 false。

bool InvertRgn(const TRegion& region);

将指定的区域颜色取反。如果成功就返回 true，否则就返回 false。

bool LineTo(int x, int y);

在 DC 中使用当前的画笔从当前的位置开始画一条线段到指定点，但不包括指定点。如果成功就返回

true，否则就返回 false。

bool LineTo(const TPoint& point);

在 DC 中使用当前的画笔从当前的位置开始画一条线段到指定点，但不包括指定点。如果成功就返回 true，否则就返回 false。

bool LPtoDP(TPoint * points, int count = 1) const;

将 points 所指的点数组中的每一个点的坐标从逻辑坐标转换成设备坐标。如果成功就返回 true，否则就返回 false。

bool MaskBlt(const TRect& dst, const TDC& srcDC, const TPoint& src, const TBitmap& maskBm, const TPoint& maskPos, uint32 rop);

从所给的 DC 中拷贝位图到此 DC 中，如果成功就返回 true，否则就返回 false。

bool ModifyWorldTransform(XFORM far& xform, uint32 mode);

改变 DC 中的 world transformation。

bool MoveTo(int x, int y);

将 DC 中当前位置设置为所给的点，如果成功就返回 true，否则就返回 false。

bool MoveTo(const TPoint& point);

将 DC 中当前位置设置为所给的点，如果成功就返回 true，否则就返回 false。

bool MoveTo(const TPoint& point, TPoint& oldPoint);

将 DC 中当前位置设置为所给的点，如果成功就返回 true，否则就返回 false。

int OffsetClipRgn(const TPoint& delta);

将剪贴区域在水平方向移动 x，在垂直方向移动 y。返回值为剪贴区域的类型，意义见表 A-64。

表 A-64

区 域	意 义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

virtual bool OffsetViewportOrg(const TPoint& delta, TPoint * oldOrg = 0);

将 DC 中视口原点在水平方向移动 x，在垂直方向移动 y。如果成功就返回 true，否则就返回 false。

bool OffsetWindowOrg(const TPoint& delta, TPoint * oldOrg = 0);

将与 DC 相关的窗口原点在水平方向移动 x，在垂直方向移动 y。如果成功返回 true，否则返回false。

void OWLFastWindowFrame(TBrush& brush, TRect& rect, int xWidth, int yWidth);

按照所给的宽度用所给的刷子画一个区域的边框。

bool PaintRgn(const TRegion& region);

在所给的区域内绘图，区域内部使用当前刷子填充。如果成功就返回 true，否则就返回 false。

bool PatBlt(int x, int y, int w, int h, uint32 rop=PATCOPY);

在 DC 中使用当前选择的刷子在所给的矩形中绘图，如果成功就返回 true，否则就返回 false。rop 参数指定的光栅操作的取值和意义如表 A-65 所示。

表 A-65

取　值	意　　义
PATCOPY	把模式拷贝到目标位图
PATINVERT	用或操作转换位图
DSTINVERT	将目标位图取反
BLACKNESS	输出全部为黑
WHITENESS	输出全部为白

bool PatBlt(const TRect& dst, uint32 rop==PATCOPY);

在 DC 中使用当前选择的刷子在所给的矩形中绘图，如果成功就返回 true，否则就返回 false。rop 参数指定的光栅操作的取值和意义如表 A-66 所示。

表 A-66

取　值	意　　义
PATCOPY	把模式拷贝到目标位图
PATINVERT	用或操作转换位图
DSTINVERT	将目标位图取反
BLACKNESS	输出全部为黑
WHITENESS	输出全部为白

HRGN PathToRegion();

如果成功就返回路径产生的区域，否则就返回 0。

bool Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

画一个饼图，包括一段椭圆弧和椭圆中心的连线。如果成功就返回 true，否则就返回 false。

bool Pie(const TRect& rect, const TPoint& start, const TPoint& end);

画一个饼图，包括一段椭圆弧和椭圆中心的连线。如果成功就返回 true，否则就返回 false。

bool PlayMetaFile(const TMetaFilePict& metaFile);

在 DC 上显示指定的图元文件。如果成功就返回 true，否则就返回 false。

```
void PlayMetaFileRecord(HANDLETABLE far& Handletable,
                        METARECORD far& metaRecord,
                        int count);
```

输出指定的图元文件记录。

```
bool PlgBlt(const TPoint& dst,
            const TDC& srcDC,
            const TRect& src,
            const TBitmap& maskBm,
            const TPoint& maskPos,
            uint32 rop=SRCCOPY);
```

(32 位应用程序)从源 DC 中传输位块到当前 DC 中。如果成功就返回 true，否则就返回 false。

bool PolyBezier(const TPoint * points, int count);

使用当前的选择画笔根据所给的点列绘制一个或者多个立体的 Bezier 曲线。

bool PolyBezierTo(const TPoint * points, int count);

使用当前的选择画笔根据所给的点列绘制一个或者多个立体的 Bezier 曲线。

bool PolyDraw(const TPoint * points, uint8 * types, int count);

使用当前的选择画笔根据所给的点列绘制图形。

bool Polygon(const TPoint * points, int count);

绘制一个自动封闭的多边形。如果成功就返回 true，否则就返回 false。

bool Polyline(const TPoint * points, int count);

在 DC 中根据所给的点列绘制多条相连线段。如果成功就返回 true，否则就返回 false。

bool PolylineTo(const TPoint * points, int count);

在 DC 中根据所给的点列绘制多条相连线段。如果成功就返回 true，否则就返回 false。

bool PolyPolygon(const TPoint * points, const int * PolyCounts, int count);

在 DC 中根据所给的点列绘制一组封闭的多边形。如果成功就返回 true，否则就返回 false。

bool PolyPolyline(const TPoint * points, const int * PolyCounts, int count);

在 DC 中根据所给的点列绘制多个相连的多线段。如果成功就返回 true，否则就返回 false。

bool PtVisible(const TPoint& point) const;

如果所给的点在 DC 的剪贴区域中就返回 true，否则就返回 false。

int RealizePalette();

将 DC 中的逻辑调色板与系统调色板对应起来。

bool Rectangle(int x1, int y1, int x2, int y2);

使用 DC 中当前选择的画笔绘制一个指定的矩形并用当前选择的刷子填充它。如果成功就返回 true，否则就返回 false。

bool Rectangle(const TPoint& p1, const TPoint& p2);

使用 DC 中当前选择的画笔绘制一个指定的矩形并用当前选择的刷子填充它。如果成功就返回 true，否则就返回 false。

bool Rectangle(const TPoint& point, const TSize& s);

使用 DC 中当前选择的画笔绘制一个指定的矩形并用当前选择的刷子填充它。如果成功就返回 true，否则就返回 false。

bool Rectangle(const TRect& rect);

使用 DC 中当前选择的画笔绘制一个指定的矩形并用当前选择的刷子填充它。如果成功就返回 true，否则就返回 false。

bool RectVisible(const TRect& rect) const;

如果所给的矩形全部位于 DC 中的剪贴区域中就返回 true，否则就返回 false。

virtual bool ResetDC(DEVMODE far& devMode);

用所给的值更新 DC 中的信息，如果成功返回 ture，否则返回 false。DEVMODE 结构的定义如下：

16 位版本的定义：

```
typedef struct tagDEVMODE {
  char dmDeviceName[CCHDEVICENAME];
  UINT dmSpecVersion;
  UINT dmDriverVersion;
  UINT dmSize;
  UINT dmDriverExtra;
  DWORD dmFields;
```

```
  int dmOrientation;
  int dmPaperSize;
  int dmPaperLength;
  int dmPaperWidth;
  int dmScale;
  int dmCopies;
  int dmDefaultSource;
  int dmPrintQuality;
  int dmColor;
  int dmDuplex;
  int dmYResolution;
  int dmTTOption;
} DEVMODE;
```

32 位版本的定义：

```
typedef struct devicemode {
  TCHAR dmDeviceName[32];
  WORD dmSpecVersion;
  WORD dmDriverVersion;
  WORD dmSize;
  WORD dmDriverExtra;
  DWORD dmFields;
  short dmOrientation;
  short dmPaperSize;
  short dmPaperLength;
  short dmPaperWidth;
  short dmScale;
  short dmCopies;
  short dmDefaultSource;
  short dmPrintQuality;
  short dmColor;
  short dmDuplex;
  short dmYResolution;
  short dmTTOption;
  short dmCollate;
  TCHAR dmFormName[32];
  WORD dmUnusedPadding;
  USHORT dmBitsPerPel;
  DWORD dmPelsWidth;
  DWORD dmPelsHeight;
  DWORD dmDisplayFlags;
  DWORD dmDisplayFrequency;
} DEVMODE;
```

void RestoreBrush();

恢复初始时的刷子设置。

virtual bool RestoreDC(int savedDC = -1);

将 DC 的状态恢复到 savedDC 指定的状态。如果成功就返回 true，否则就返回 false。

virtual void RestoreFont();

恢复初始时的字体设置。

void RestoreObjects();

恢复初始时的全部对象设置。

void RestorePalette();

恢复初始时的调色板设置。

void RestorePen();

恢复初始时的画笔设置。

void RestoreTextBrush();

恢复初始时的文本刷子设置。

bool RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);

绘制一个圆角矩形。如果成功返回 true，否则就返回 false。

bool RoundRect(const TPoint& p1, const TPoint& p2, const TPoint& rad);

绘制一个圆角矩形。如果成功就返回 true，否则就返回 false。

bool RoundRect(const TPoint& p, const TSize& s, const TPoint& rad);

绘制一个圆角矩形。如果成功就返回 true，否则就返回 false。

bool RoundRect(const TRect& rect, const TPoint& rad);

绘制一个圆角矩形。如果成功就返回 true，否则就返回 false。

virtual int SaveDC() const;

将 DC 的信息保存在设备上下文栈中。

```
virtual bool ScaleViewportExt(int xNum, int xDenom,
                              int yNum, int yDenom,
                              TSize * oldExtent = 0);
```

改变当前视口的长度值，如果成功就返回 true，否则就返回 false。

```
virtual bool ScaleWindowExt(int xNum, int xDenom,
                            int yNum, int yDenom,
                            TSize * oldExtent = 0);
```

改变当前窗口的长度值。如果成功就返回 true，否则就返回 false。

```
bool ScrollDC(int x, int y,
              const TRect& scroll,
              const TRect& clip,
              TRegion& updateRgn,
              TRect& updateRect);
```

按照所给的滚动距离在水平和垂直方向滚动一个矩形，如果成功就返回 true，否则就返回 false。

```
bool ScrollDC(const TPoint& delta,
              const TRect& scroll,
              const TRect& clip,
              TRegion& updateRgn,
              TRect& updateRect);
```

按照所给的滚动距离在水平和垂直方向滚动一个矩形，如果成功就返回 true，否则就返回 false。

bool SelectClipPath(int mode);

选择当前的路径为剪贴区域，如果成功就返回 true，否则就返回 false。

int SelectClipRgn(const TRegion& region);

选择所给的区域为剪贴区域。返回值为剪贴区域的类型，见表 A-67。

void SelectObject(const TBrush& brush);

在 DC 中选择当前的刷子对象。

void SelectObject(const TPen& pen);

在 DC 中选择当前的画笔对象。

表 A-67

区　　域	意　　　义
COMPLEXREGION	剪贴区域具有覆盖边框
ERROR	失效的 DC
NULLREGION	剪贴区域为空
SIMPLEREGION	剪贴区域不具有覆盖边框

virtual void SelectObject(const TFont& font);

在 DC 中选择当前的字体对象。

void SelectObject(const TPalette& palette, bool forceBackground = false);

在 DC 中选择当前的调色板对象。

virtual void SelectStockObject(int index);

在 DC 中选择预定义的对象。

virtual TColor SetBkColor(TColor color);

设置 DC 中的背景颜色为所给的颜色。

int SetBkMode(int mode);

设置 DC 中的背景模式为所给的模式，它的值是 OPAQUE 或者 TRANSPARENT。

uint SetBoundsRect(TRect& bounds, uint flags);

设置 DC 的限定矩形为所给的矩形。返回的限定矩形状态的取值和意义如表 A-68 所示。

表 A-68

取　　值	意　　　义
DCB_ACCUMULATE	进行限定矩形的累积
DCB_DISABLE	限定矩形的累积被禁止
DCB_ENABLE	限定矩形的累积被激活
DCB_RESET	限定矩形为空
DCB_SET	设置限定矩形区域

bool SetBrushOrg(const TPoint& origin, TPoint * oldOrg = 0);

定义 DC 中当前刷子的原点坐标为所给的坐标，如果成功就返回 true，否则就返回 false。

bool SetDIBits(TBitmap& bitmap,
　　　　uint16 startScan,

```
        uint16 numScans,
        const void HUGE * bits,
        const BITMAPINFO far& Info,
        uint16 usage);
```

根据 BITMAPINFO 结构中的值设置一个与设备无关位图的像素点的值，如果成功就返回 true，否则就返回 false。BITMAPINFO 结构的定义如下：

```
typedef struct tagBITMAPINFO
{
  BITMAPINFOHEADER bmiHeader;
  RGBQUAD bmiColors[1];
} BITMAPINFO;
```

```
bool SetDIBits(TBitmap& Bitmap, const TDib& dib);
```

设置一个与设备无关位图的像素点的值，如果成功就返回 true，否则就返回 false。

```
bool SetDIBitsToDevice(const TRect& dst,
                       const TPoint& src,
                       uint16 startScan,
                       uint16 numScans,
                       const void HUGE * bits,
                       const BITMAPINFO far& bitsInfo,
                       uint16 usage);
```

直接在 DC 所指的设备上显示一个与设备无关的位图，如果成功就返回 true，否则就返回 false。BITMAPINFO 结构的定义如下：

```
typedef struct tagBITMAPINFO
{
  BITMAPINFOHEADER bmiHeader;
  RGBQUAD bmiColors[1];
} BITMAPINFO;
```

```
bool SetDIBitsToDevice(const TRect& dst, const TPoint& src, const TDib& dib);
```

直接在 DC 所指的设备上显示一个与设备无关的位图。如果成功就返回 true，否则就返回 false。

```
virtual int SetMapMode(int mode);
```

定义 DC 的映射模式。mode 取值和意义如表 A－69 所示。

表 A－69

取　值	意　义
MM_ANISOTROPIC	X 和 Y 为独立标度的任意单位
MM_HIENGLISH	每一个逻辑单位为 0.001 英寸。坐标原点在窗口的左下角
MM_HIMETRIC	每一个逻辑单位为 0.01 毫米。坐标原点在窗口的左下角
MM_ISOTROPIC	X 和 Y 为统一标度的任意单位
MM_LOENGLISH	每一个逻辑单位为 0.01 英寸。坐标原点在窗口的左下角
MM_LOMETRIC	每一个逻辑单位为 0.1 毫米。坐标原点在窗口的左下角
MM_TEXT	每一个逻辑单位一个物理像素点。坐标原点在窗口的左上角
MM_TWIPS	每一个逻辑单位为 1/1440 英寸。坐标原点在窗口的左下角

uint32 SetMapperFlags(uint32 flag);

设置逻辑字体到物理字体的映射算法。

bool SetMiterLimit(float newLimit, float * oldLimit = 0);

设置新的的 MiterLimit。如果成功就返回 true，否则就返回 false。

TColor SetPixel(int x, int y, TColor color);

设置像素点的颜色值。返回原来的颜色值。

TColor SetPixel(const TPoint& p, TColor color);

设置像素点的颜色值。返回原来的颜色值。

int SetPolyFillMode(int mode);

设置多边形填充模式。mode 的取值是 ALTERNATE 或者 WINDING。返回以前的值。

int SetROP2(int mode);

设置 DC 中的绘图模式。mode 的取值和意义如表 A-70 所示。返回以前的值。

表 A-70

取　值	意　义
R2_BLACK	像素点总为黑色
R2_COPYPEN	像素点为画笔颜色
R2_MASKNOTPEN	像素点为显示颜色与反画笔颜色的与
R2_MASKPEN	像素点为显示颜色与画笔颜色的与
R2_MASKPENNOT	像素点为反显示颜色与画笔颜色的与
R2_MERGEPEN	像素点为显示颜色与画笔颜色的或
R2_MERGENOTPEN	像素点为显示颜色与反画笔颜色的或
R2_MERGEPENNOT	像素点为反显示颜色与画笔颜色的或
R2_NOP	像素点保持不变
R2_NOT	像素点是显示颜色的反
R2_NOTCOPYPEN	像素点是画笔颜色的反
R2_NOTMASKPEN	像素点是 R2_MASKPEN 值的反
R2_NOTMERGEPEN	像素点是 R2_MERGEPEN 值的反
R2_NOTXORPEN	像素点是 R2_XORPEN 值的反
R2_WHITE	像素点总是白色
R2_XORPEN	像素点是显示颜色和画笔颜色的异或

int SetStretchBltMode(int mode);

设置 DC 的缩放模式，返回以前的值。

int SetSystemPaletteUse(int usage);

设置系统调色板用法。

uint SetTextAlign(uint flags);

设置 DC 中文本的对齐方式。flags 的取值和意义如表 A-71 所示。返回以前的值。

表 A-71

取　值	意　义
TA_BASELINE	限定的矩形内字体的基线与 x 轴对齐
TA_BOTTOM	限定矩形的底边与 x 轴对齐
TA_TOP	限定矩形的顶部与 x 轴对齐
TA_CENTER	y 轴与限定矩形的中心对齐
TA_LEFT	y 轴与限定矩形的左边对齐
TA_RIGHT	y 轴与限定矩形的右边对齐
TA_NOUPDATECP	不更新当前的位置
TA_UPDATECP	更新当前位置

int SetTextCharacterExtra(int extra);

设置文本的字符间距，返回以前的值。

virtual TColor SetTextColor(TColor color);

设置文本的颜色，返回以前的值。

bool SetTextJustification(int breakExtra, int breakCount);

用所给的参数设置一个文本行，如果成功就返回 true，否则就返回 false。

virtual bool SetViewportExt(const TSize& extent, TSize * oldExtent = 0);

设置视口的宽度和高度，如果成功就返回 true，否则就返回 false。

virtual bool SetViewportOrg(const TPoint& origin, TPoint * oldOrg = 0);

设置视口的原点坐标，如果成功就返回 true，否则就返回 false。

virtual bool SetWindowExt(const TSize& extent, TSize * oldExtent = 0);

设置窗口的宽度和高度，如果成功就返回 true，否则就返回 false。

bool SetWindowOrg(const TPoint& origin, TPoint * oldOrg = 0);

设置窗口的原点坐标。如果成功就返回 true，否则就返回 false。

bool SetWorldTransform(XFORM far& xform);

设置 WorldTransform 的值。如果成功就返回 true，否则就返回 false。

```
bool StretchBlt(int dstX, int dstY,
                int dstW, int dstH,
                const TDC& srcDC,
                int srcX, int srcY,
                int srcW, int srcH,
                uint32 rop=SRCCOPY);
```

将位图从源矩形区域中读出并经过一定的缩放填入目标矩形，如果成功就返回 true，否则就返回 false。

```
bool StretchBlt(const TRect& dst,
                const TDC& srcDC,
                const TRect& src,
                uint32 rop=SRCCOPY);
```

将位图从源矩形区域中读出并经过一定的缩放填入目标矩形，如果成功就返回 true，否则就返回 false。

```
bool StretchDIBits(const TRect& dst,
                   const TRect& src,
                   const void HUGE * bits,
                   const BITMAPINFO far& bitsInfo,
                   uint16 usage,
                   uint32 rop=SRCCOPY);
```

将一个与设备无关的位图从源矩形区域中读出并经过一定的缩放填入目标矩形，如果成功就返回 true，否则就返回 false。

```
bool StretchDIBits(const TRect& dst,
                   const TRect& src,
                   const TDib& dib,
                   uint32 rop=SRCCOPY);
```

将一个与设备无关的位图从源矩形区域中读出并经过一定的缩放填入目标矩形，如果成功就返回 true，否则就返回 false。

```
bool StrokeAndFillPath();
```

（用于 32 位应用程序）填充当前的路径，如果成功就返回 true，否则就返回 false。

```
bool StrokePath();
```

（用于 32 位应用程序）提高当前的路径使用当前的画笔，如果成功就返回 true，否则就返回 false。

```
bool TabbedTextOut(const TPoint& p,
                   const char far * string,
                   int count,
                   int numPositions,
                   const int far * positions,
                   int tabOrigin);
```

在 tabOrigin 指定的位置处用当前选择的字体显示一个字符串，如果成功返回 true，否则返回false。

```
virtual bool TabbedTextOut(const TPoint& p,
                   const char far * string,
                   int count,
                   int numPositions,
                   const int far * positions,
                   int tabOrigin,
                   TSize& size);
```

在 tabOrigin 指定的位置处用当前选择的字体显示一个字符串，如果成功返回 true，否则返回false。

```
virtual bool TextOut(int x, int y, const char far * string, int count = -1);
```

在 DC 中使用当前的字体显示字符串，如果成功就返回 true，否则就返回 false。

```
bool TextOut(const TPoint& p, const char far * string, int count = -1);
```

在 DC 中使用当前的字体显示字符串，如果成功就返回 true，否则就返回 false。

```
bool TextRect(int x1, int y1, int x2, int y2);
```

填充指定的矩形，将剪贴的字符串填入区域中，如果成功就返回 true，否则就返回 false。

```
bool TextRect(const TRect& rect);
```

填充指定的矩形，将剪贴的字符串填入区域中，如果成功就返回 true，否则就返回 false。

```
bool TextRect(int x1, int y1, int x2, int y2, TColor color);
```

填充指定的矩形，将剪贴的字符串填入区域中，如果成功就返回 true，否则就返回 false。

bool TextRect(const TRect rect, TColor color);

填充指定的矩形，将剪贴的字符串填入区域中，如果成功就返回 true，否则就返回 false。

void UpdateColors();

更新 DC 中的用户区域，把用户区域中的当前颜色匹配到系统调色板中的颜色。

bool WidenPath();

(用于 32 位应用程序)重新设置 DC 中的路径。

Protected 构造函数：

TDC();

TDC(HDC handle, TAutoDelete AutoDelete);

Protected 数据成员：

TGDIBase::Handle;

DC 的句柄。

HBRUSH OrgBrush;

初始刷子句柄。

HFONT OrgFont;

初始字体句柄。

HPALETTE OrgPalette;

初始调色板句柄。

HPEN OrgPen;

初始画笔句柄。

HBRUSH OrgTextBrush;

初始文本刷子句柄。

TGdiBase::ShouldDelete;

指明是否自动删除句柄。

Protected 成员函数：

TGdiBase::CheckValid(uint resId=IDS_GDIFAILURE);

检查 GDI 对象句柄是否有效。

static void CheckValid(HANDLE handle, uint resId=IDS_GDIFAILURE);

检查 GDI 对象句柄是否有效。

virtual HDC GetAttributeHDC() const;

返回 GDI 对象的属性。

HDC GetHDC() const;

返回 DC 的句柄。

void Init();

初始化 DC。

A. 60 TWindowDC 类

头文件：dc. h

描述：作为 TDC 的派生类，TWindowDC 提供了可以访问一个窗口的全部区域的设备上下文。TWindowDC 类和它的派生类在使用结束后都释放句柄，它在 OWL 5. 0 中的位置如图 A－52 所示。

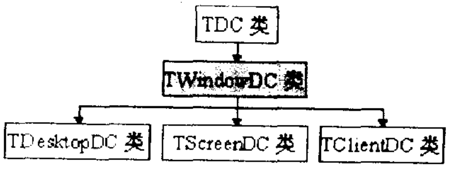


图 A－52 TWindowDC 类在 OWL 5. 0 中的位置

Public 构造函数和析构函数：

TWindowDC(HWND wnd)；

~TWindowDC()；

Protected 构造函数：

TWindowDC()；

Protected 数据成员：

HWND Wnd；

保存 DC 所属的窗口句柄。

A. 61 TDesktopDC 类

头文件：dc. h

描述：作为 TWindowDC 的派生类，TDesktopDC 提供了访问 Windows 的 desktop 窗口的设备上下文。TDesktopDC 类在 OWL 5. 0 中的位置如图 A－53 所示。

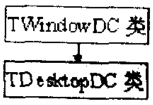


图 A－53 TDesktopDC 类在 OWL 5. 0 中的位置

Public 构造函数：

TDesktopDC()；

A. 62 TScreenDC 类

头文件：dc. h

描述：作为 TWindowDC 的派生类，TScreenDC 提供了直接访问屏幕位图的设备上下文。TScreenDC 中的设备上下文句柄的值为 0，它包括了整个屏幕。TScreenDC 类在 OWL 5. 0 中的位置如图 A－54 所示。

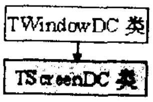


图 A－54 TScreenDC 类在 OWL 5. 0 中的位置

Public 构造函数：

TScreenDC()；

A. 63 TClientDC 类

头文件：dc. h

描述：作为 TWindowDC 的派生类，TClientDC 提供了访问一个窗口中客户区域的设备上下文，它在 OWL 5. 0 中的位置如图 A－55 所示。

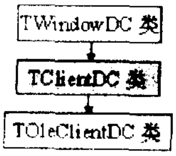


图 A－55 TClientDC 类在 OWL 5. 0 中的位置

Public 构造函数：

TClientDC(HWND wnd)；

A. 64 TOleClientDC 类

头文件：olewindo. h

描述：作为 TClientDC 的派生类，TOleClientDC 提供了在两个不同坐标系中转换的功能，它在 OWL 5. 0 中的位置如图 A－56 所示。

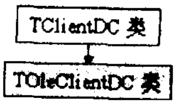


图 A－56 TOleclientDC 类在 OWL 5. 0 中的位置

Public 构造函数：

TOleClientDC(TOleWindow& win，bool scale = true)；

A. 65 TPaintDC 类

头文件：dc. h

描述：作为 TWindowDC 的派生类，TPaintDC 类在响应 WM_PAINT 时提供设备上下文，它在 OWL 5. 0 中的位置如图 A－57 所示。

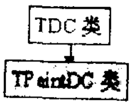


图 A－57 TPaintDC 类在 OWL 5. 0 中的位置

Public 数据成员：

PAINTSTRUCT Ps；

Public 构造函数和析构函数：

TPaintDC(HWND wnd)；

~TPaintDC()；

Protected 数据成员：

HWND Wnd；

A. 66 TMetaFileDC 类

头文件：dc. h

描述：作为 TDC 的派生类，TMetaFileDC 提供了访问一个图元文件的设备上下文，它在 OWL 5. 0 中的

位置如图 A-58 所示。

Public 构造函数和析构函数：

```
TMetaFileDC(const char far * filename = 0);
~TMetaFileDC();
```

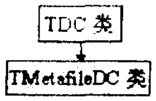


图 A-58　TMetaFileDC 类在 OWL 5.0 中的位置

Public 成员函数：

```
HMETAFILE Close();
```

A.67　TCreatedDC 类

头文件： dc.h

描述： 作为一个抽象的 TDC 类，TCreatedDC 具有创建和删除 DC 的功能，它在 OWL 5.0 中的位置如图 A-59 所示。

Public 构造函数和析构函数：

```
TCreatedDC(const char far * driver,
           const char far * device,
           const char far * output,
           const DEVMODE far * initData=0);
TCreatedDC(HDC handle TAutoDelete autoDelete);
~TCreatedDC();
```

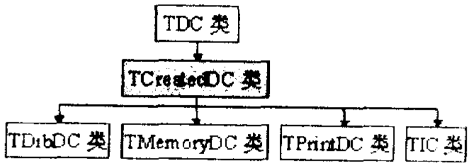


图 A-59　TCreatDC 类在 OWL 5.0 中的位置

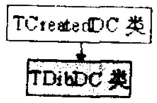
Protected 构造函数：

```
TCreatedDC();
```

A.68　TDibDC 类

头文件： dc.h

描述： 作为 TDC 的派生类，TDibDC 提供了访问与设备无关位图的设备上下文，它在 OWL 5.0 中的位置如图 A-60 所示。



Public 构造函数：

```
TDibDC(const TDib& dib);
```

图 A-60　TDibDC 类在 OWL 5.0 中的位置

A.69　TPrintDC 类

头文件： dc.h

描述： 作为 TDC 的派生类，TPrintDC 提供访问一个打印机的设备上下文，它在 OWL 5.0 中的位置如图 A-61 所示。

Public 构造函数：

```
TPrintDC(HDC handle, TAutoDelete autoDelete = NoAutoDelete);
TPrintDC(const char far * driver,
         const char far * device,
         const char far * output,
         const DEVMODE far * initData);
```

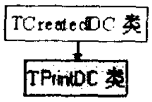


图 A-61 TPrintDC 类在 OWL 5.0 中的位置

Public 成员函数：

int AbortDoc();

退出当前的打印任务。

int BandInfo(TBandInfo& bandInfo);

返回设备结合信息。

```
static uint32 DeviceCapabilities(const char far * driver,
              const char far * device,
              const char far * port,
              int capability,
              char far * output=0,
              LPDEVMODE devmode=0);
```

获得设备的信息。

int EndDoc();

结束当前的打印任务。

int EndPage();

结束当前的打印页。

int Escape(int escape, int count=0, const void * inData=0, void * outData=0);

可存取 GDI 不能存取的设备。

int NextBand(TRect& rect);

下一个 Band。

bool QueryAbort(int rsvd=0);

(用于 16 位应用程序)询问是否结束了一个打印任务。

uint QueryEscSupport(int escapeNum);

如果在设备中定义了 escapeNum，就返回 true，否则就返回 false。

int SetAbortProc(ABORTPROC proc);

被 AbortDoc 函数调用，用于结束一个打印任务。

int SetCopyCount(int reqestCount, int& actualCount);

设置 requestCount 的值，如果成功就返回 1，否则返回 0。

int StartDoc(const char far * docName, const char far * output);

开始一个打印任务。

int StartPage();

让设备准备接收数据。

Protected 数据成员：

DOCINFO DocInfo;

A.70　TIC 类

头文件：dc.h

描述：作为 TDC 的派生类，TIC 类提供了一个构造函数从一个驱动器、设备和端口创建一个 DC 对象，它在 OWL 5.0中的位置如图 A－62 所示。

Public 构造函数：

```
TIC(const char far * driver,
    const char far * device,
    const char far * output,
    const DEVMODE far * initData=0);
```

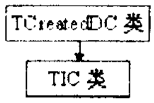


图 A－62　TIC 类在 OWL 5.0 中的位置

A.71　TMemoryDC 类

头文件：dc.h

描述：作为 TDC 的派生类，TMemoryDC 可以构造访问一个内存 DC 的对象，它在 OWL 5.0 中的位置如图A－63所示。

Public 构造函数：

```
TMemoryDC();
TMemoryDC(const TDC& DC);
TMemoryDC(HDC handle, TAutoDelete autoDelete);
```

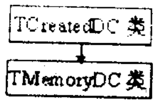


图 A－63　TMemory 类在 OWL 5.0 中的位置

Public 成员函数：

void RestoreBitmap();

恢复一个初始时的位图。

void RestoreObjects();

void SelectObject(const TBrush& brush);

void SelectObject(const TPen& pen);

void SelectObject(const TFont& font);

void SelectObject(const TPalette& palette, bool forceBackground=false);

void SelectObject(const TBitmap& bitmap);

以上均是恢复初始选择对象的函数。

Protected 数据成员：

HBITMAP OrgBitmap;

A.72　TColor 类

头文件：color.h

描述：TColor 类提供了一些对颜色的操作，它包括了十个代表标准的 RGB 颜色值的静态数据成员。

Public 数据成员：

static const TColor Black;

具有固定的 RGB 值，代表颜色 Black。

static const TColor Gray;

具有固定的 RGB 值，代表颜色 Gray。

static const TColor LtBlue;

具有固定的 RGB 值，代表颜色 LtBlue。

static const TColor LtCyan;

具有固定的 RGB 值，代表颜色 LtCyan。

static const TColor LtGray;

具有固定的 RGB 值，代表颜色 LtGray。

static const TColor LtGreen;

具有固定的 RGB 值，代表颜色 LtGreen。

static const TColor LtMagenta;

具有固定的 RGB 值，代表颜色 LtMagenta。

static const TColor LtRed;

具有固定的 RGB 值，代表颜色 LtRed。

static const TColor LtYellow;

具有固定的 RGB 值，代表颜色 LtYellow。

static const TColor White;

具有固定的 RGB 值，代表颜色 White。

Public 构造函数：

TColor();

TColor(COLORREF value);

TColor(long value);

TColor(int r int g int b);

TColor(int r int g int b int f);

TColor(int index);

TColor(const PALETTEENTRY far& pe);

TColor(const RGBQUAD far& q);

TColor(const RGBTRIPLE far& t);

Public 成员函数：

uint8 Blue() const;

返回颜色的蓝色部分。

uint8 Flags() const;

返回对象的 peFlags 值。

static TColor GetSysColor(int uiElement);

返回系统的颜色值。

uint8 Green() const;

返回颜色的绿色部分。

bool operator ==(const TColor& clrVal)const;

如果所给的颜色与此颜色系统就返回 true。

operator COLORREF() const;

类型转换操作，返回 Value 的值。

int Index() const;

返回颜色的索引。

TColor PalIndex() const;

返回颜色在调色板中的索引。

TColor PalRelative() const;

返回与调色板对应的 RGB 值。

uint8 Red() const;

返回颜色的红色部分。

uint32 Rgb() const;

获得颜色的 RGB 值。

static bool SetSysColors(unsigned nelems, const int indices[], const TColor colors[]);

设置 UI element 颜色。

Protected 数据成员:

COLORREF Value;

A.73　TLayoutWindow 类

头文件: layoutwi.h

描述: 作为 TWindow 的派生类，TLayoutWindow 提供了定义一个窗口的布局规则。调用它的构造函数，你可以创建一个与另外一个窗口的大小和位置相适应的窗口，常用的工具条和状态条就是这样的窗口。

TLayout Window 类在 OWL 5.0 中的位置如图 A-64 所示。

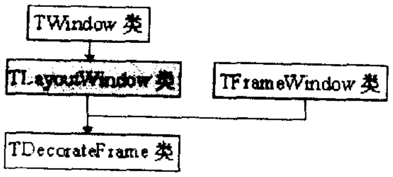


图 A-64　TLayoutWindow 类在 OWL 5.0 中的位置

Public 构造函数和析构函数:

```
TLayoutWindow(TWindow * parent,
            const char far * title = 0,
            TModule * module = 0);
~TLayoutWindow();
```

Public 成员函数:

bool GetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);

获得子窗口的布局规则。

void Layout();

让窗口调整子窗口的大小和位置以适应特殊的规则。当布局规则改变时你可以调用 Layout 函数。

bool RemoveChildLayoutMetrics(TWindow &child);

取消一个子窗口的布局规则。

void SetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);

给窗口设置布局规则，替换原来的规则。

例如，设置如下规则：

```
layoutMetrics->X.Absolute(lmLeft, 10);
layoutMetrics->Y.Absolute(lmTop, 10);
layoutMetrics->Width.Set(lmWidth, lmRightOf, GetClientWindow(), lmWidth, -40);
layoutMetrics->Height.Set(lmHeight, lmRightOf, GetClientWindow(), lmHeight, -40);
```

然后调用 SetChildLayoutMetrics 来设置子窗口：

```
SetChildLayoutMetrics(*MyChildWindow, *layoutMetrics);
```

Protected 数据成员：

TSize ClientSize;

客户窗口的大小。

Protected 成员函数：

void EvSize(uint sizeType, TSize& size);

响应改变窗口大小的事件。

事件响应表：见表 A-72。

表 A-72

EV_ macros	成员函数
EV_WM_SIZEEvSize	

A.74　TLayoutConstraint 结构

头文件：layoutco.h

描述：TLayoutConstraint 结构定义了一个窗口的边界(或大小)与另外一个窗口(父窗口或兄弟窗口)的边界(或大小)的关系。如果两个窗口是父子关系，子窗口的尺寸就取决于父窗口。同一窗口的尺寸间也可以有互相决定的关系，比如窗口的宽可以总是长的两倍。TLayoutMetrics 类可列出这些尺寸和边界的规则。

Public 数据成员：

uint MyEdge;

MyEdge 包含了窗口边界或大小的方式，包括，lmTop、lmBottom、lmLeft、lmRight、lmCenter、lmWidth 和 lmHeight。

uint OtherEdge;

OtherEdge 包含了其它窗口边界或大小的方式，包括，lmTop、lmBottom、lmLeft、lmRight、lmCenter、lmWidth 和 lmHeight。

TRelationship Relationship;

Relationship 确定两个窗口的已存在的关系类型，包括，lmRightOf、lmLeftOf、lmAbove、lmBelow、lmSameAs 和 lmPercentOf。

TWindow *RelWin;

RelWin 是一个指向兄弟窗口的指针，当窗口的尺寸是父窗口决定的，它的值为 lmParent。当窗口中的

一些尺寸由另外一些尺寸决定时(比如窗口的宽可以总是长的两倍)，RelWin 指向窗口自己。

TMeasurementUnits Units;

Units 枚举了测量单位。

```
enum TMeasurementUnits{lmPixels, lmLayoutUnits};
union {
    int Margin;
    int Value;
    int Percent;
    };
```

A.75 TLayoutMetrics 类

头文件：layoutwi.h

描述：TLayoutMetrice 类包括四个用于定义窗口的布局规则的布局约束。表 A-73 给出了可以在 X、Y、Height 和 Width 字段中的取值。

表 A-73

字　段	取　　值
X	lmLeft, lmCenter, lmRight
Y	lmTop, lmCenter, lmBottom
Height	lmCenter, lmRight, lmWidth
Width	lmCenter, lmBottom, lmHeight

如果子窗口的布局规则是由父窗口决定的，它的关系窗口指针应该是 lmParent(而不是父窗口的指针，也不能是 this)，例如：

```
TWindow * child = new TWindow(this, "");
TLayoutMetrics metrics;
metrics.X.Set(lmCenter, lmSameAs, lm()t, lmCenter);
metrics.Y.Set(lmCenter, lmSameAs, lm()t, lmCenter);
SetChildLayoutMetrics( * child, metrics);
```

Public 数据成员：

TEdgeOrWidthConstraint Height;

Height 包含了窗口的高度约束、中间边界约束或底边界约束。

TEdgeOrWidthConstraint Width;

Width 包含了窗口的宽度约束、中间边界约束或右边界约束(lmRight)。

TEdgeConstraint X, Y;

X 包含了窗口的 X (left, center, right)边的约束;Y 包含了窗口的 Y (top, center, bottom)边的约束。

Public 构造函数：

TLayoutMetrics();

创建一个 TLayoutMetrics 对象并初始化这个对象。TLayoutMetrics 设置下列的值：

```
X.RelWin = 0;
X.MyEdge = lmLeft;
X.Relationship = lmAsIs;
X.Units = lmLayoutUnits;
X.Value = 0;
Y.RelWin = 0;
Y.MyEdge = lmTop;
Y.Relationship = lmAsIs;
Y.Units = lmLayoutUnits;
Y.Value = 0;
Width.RelWin = 0;
Width.MyEdge = lmWidth;
Width.Relationship = lmAsIs;
Width.Units = lmLayoutUnits;
Width.Value = 0;
Height.RelWin = 0;
Height.MyEdge = lmHeight;
Height.Relationship = lmAsIs;
Height.Units = lmLayoutUnits;
Height.Value = 0;
```

A.76　TMDIClient 类

头文件：mdi.h

描述：TMDIclient 类是多文档界面(MDI)的客户窗口类，管理着由 TMDIFrame 建立的框架窗口的 MDI 子窗口。TMDIClient 是可流类，它在 OWL 5.0 中的位置如图 A－65 所示。

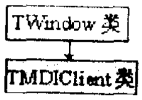


图 A－65　TMDIClient 类在 OWL 5.0 中的位置

Public 数据成员：

LPCLIENTCREATESTRUCT ClientAttr;

MDI 界面客户窗口的属性结构指针。

Public 构造函数和析构函数：

TMDIClient(TModule * module = 0);

~TMDIClient();

Public 成员函数：

virtual void ArrangeIcons();

在 MDI 界面客户窗口的底部排列子窗口的图标。

virtual void CascadeChildren();

在 MDI 界面客户窗口中设置所有非图标状态的 MDI 子窗口的大小并对这些子窗口进行排列。子窗口互相层叠，它们的标题条都是可见的。

virtual bool CloseChildren();

关闭所有的子窗口。首先调用 MDI 界面客户窗口中每一个 MDI 子窗口的 CanClose 函数，当所有的子

窗口都关闭时返回 true，否则返回 false。

bool Create()；

创建与 MDI 客户窗口相联系的界面元素。

virtual TWindow * CreateChild()；

重载 TWindow 中的同名函数，构造和创建一个新的 MDI 子窗口，并返回一个新的 MDI 子窗口指针。

TMDIChild * GetActiveMDIChild()；

GetActiveMDIChild 指向 TMDIClient 中的活动 MDI 子窗口。

virtual TMDIChild * InitChild()；

构造一个 TWindow 实例作为一个 MDI 子窗口并返回它的指针。子窗口的父窗口为 MDI 客户窗口。在你的派生 MDI 窗口类中重定义这个成员函数来创建一个派生的 MDI 子窗口类。比如：

```
PTWindowsObject TMyMDIClient::InitChild()
{
    return new TMyMDIChild(this, "");
}
```

bool PreProcessMsg(MSG &msg)；

如果接收到 WM_KEYDOWN 或 WM_SYSKEYDOWN 消息，响应 MDI 客户窗口加速键。

virtual void TileChildren(int tile = MDITILE_VERTICAL)；

在 MDI 界面客户窗口中设置所有的非图标状态的 MDI 子窗口的大小并对这些子窗口进行排列。子窗口互相不层叠，它们填满全部客户窗口。

Protected 成员函数：

void CmArrangeIcons()；

调用 ArrangeIcons 函数来响应具有 ID 值 CM_ARRANGEICONS 的菜单选择命令。

void CmCascadeChildren()；

调用 CascadeChildren 函数来响应具有 ID 值 CM_CASCADECHILDREN 的菜单选择命令。

void CmChildActionEnable(TCommandEnabler& commandEnabler)；

如果有 MDI 子窗口，CmChildActionEnalbe 打开一项子窗口行为的菜单项功能。

void CmCloseChildren()；

调用 CloseChildren 函数来响应具有 ID 值 CM_CLOSECHILDREN 的菜单选择命令。

void CmCreateChild()；

调用 CreateChild 函数来响应具有 ID 值 CM_CREATECHILD 的菜单选择命令。

voidCmTileChildren()；

调用 TileChildren 函数来响应具有 ID 值 CM_TILECHILDREN 的菜单选择命令。

void CmTileChildrenHoriz()；

调用 TileChildren 函数来响应具有 ID 值 CM_TILECHILDREN 的菜单选择命令，设置子窗口 tile 标志为 MDITILE_HORIZONTAL。

LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct)；

当子窗口创建时截获 WM_MDICREATE。如果客户窗口的风格包括 MDIS_ALLCHILDSTYLES，并且子窗口的风格是 0，就将子窗口的风格改变为 WS_VISIBLE，WS_CHILD，WS_CLIPSIBLINGS，WS_CLIPCHILDREN，WS_SYSMENU，WS_CAPTION，WS_THICKFRAME，WS_MINIMIZEBOX 和 WS_MAXIMIZEBOX。

void EvMDIDestroy(HWND hWnd)；

截获 WM_MDIDESTROY 消息。

char far * GetClassName();

返回 TMDIClient 的注册类名"MDICLIENT"。

事件响应表：见表 A－74。

表 A－74

EV_ macros	成员函数
EV_COMMAND (CM_ARRANGEICONS, CmArrangeIcons)	CmArrangeIcons
EV_COMMAND (CM_CASCADECHILDREN, CmCascadeChildren)	CmCascadeChildren
EV_COMMAND (CM_CLOSECHILDREN, CmCloseChildren)	CmCloseChildren
EV_COMMAND (CM_CREATECHILD, CmCreateChild)	CmCreateChild
EV_COMMAND (CM_TILECHILDREN, CmTileChildren)	CmTileChildren
EV_COMMAND(CM_TILECHILDRENHORIZ, CmTileChildrenHoriz)	CmTileChildrenHoriz
EV_COMMAND_ENABLE(CM_TILECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_CASCADECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_ARRANGEICONS, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_CLOSECHILDREN, CmChildActionEnable)	CmChildActionEnalbe
EV_WM_MDICREATE	EvMDICreate
EV_WM_MDIDESTROY	EvMDIDestroy

A.77　TMDIFrame 类

头文件：mdi.h

描述：TMDIFrame 是多文档界面(MDI)的框架窗口，作为 MDI 应用程序的主窗口。TMDIFrame 对象自动处理创建和初始化 MDI 客户窗口(TMDIClient 对象)，缺省时，TMDIFrame 设置窗口风格 WS_CLIPCHILDREN，这样在刷新 MDI 框架窗口和主窗口背景时就不会出现明显的闪烁。TMDIFrame 是可流类，它在 OWL 5.0 中的位置如图 A－66 所示。

因为 TMDIFrame 是 TFrameWindow 的派生类，所以它继承了键盘导航功能，这样所有的子窗口都获得了键盘导航，但是，最后只让那些真正需要键盘导航功能的子窗口打开这个功能。

想要创建一个具有 OLE 功能的 MDI 框架窗口应使用 TOleMDIFrame 类，它是 TMDIFrame 和 TOleFrame 的派生类。

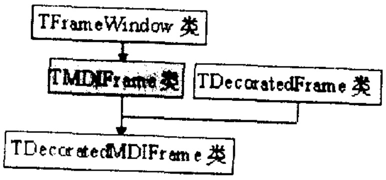


图 A－66　TMDIFrame 类在 OWL 5.0 中的位置

Public 构造函数：

```
TMDIFrame(const char far * title,
          TResId menuResId,
          TMDIClient &clientWnd = * new TMDIClient,
          TModule * module = 0)
TMDIFrame(HWND hWindow,
```

```
HWND clientHWnd,
TModule * module = 0);
```

Public 成员函数：

static HMENU FindChildMenu (HMENU newMenu);

FindChildMenu 函数用来定位菜单。这个函数从左到右查找 newMenu 菜单资源指定的弹出子菜单，这些子菜单包含了 CM_CASCADECHILDREN，CM_TITLECHILDREN 或 CM_ARRANGEICONS 命令。FindChildMenu 函数的返回值是包含这些命令之一的子菜单句柄 HMENU；如果没有找到这样的命令，FindChildMenu 函数就返回 0。

TMDIClient * GetClientWindow ();

返回客户窗口的指针。

virtual HWND GetCommandTarget();

定位并返回 command 和 command enabling 消息的目标子窗口。如果当前应用程序没有焦点或焦点在应用程序的一个工具条上，GetCommandTarget 就返回最接近活动子窗口。

void PerformCreate (int menuOrId);

重载 TWindow 的同名函数。创建 MDI 框架窗口的相关界面元素。

bool SetMenu (HMENU);

在新的菜单条中寻找 MDI 子菜单，重新设置子菜单的位置，然后给客户窗口发出一个 WM_MDISETMENU 消息设置新的菜单并调用 TWindow::DrawMenuBar 重画菜单条。如果 MDI 客户窗口表明没有以前的菜单则返回 false，否则返回 true。

Protected 成员函数：

LRESULT DefWindowProc (uint message, WPARAM wParam, LPARAM lParam);

重载 TWindow 的同名函数。

A.78　TMDIChild 类

头文件：mdichild.h

描述：TMDIChild 类定义了 MDI 子窗口的基本行为。子窗口可以在父窗口的客户区域中创建。因为子窗口存在于父窗口中并且被限制在父窗口的边界中，所以在定义子窗口之前要定义父窗口。比如，对话框可以包含子窗口，这些子窗口通常是对话框的控件对象。

要使用 MDI 子窗口，子窗口类一定要是 TMDIChild 类的派生类。MDI 子窗口从 TFrameWindow 类中继承键盘导航、焦点处理和图标设置功能。

TMDIChild 类是可流类，它在 OWL 5.0 中的位置如图 A－67 所示。

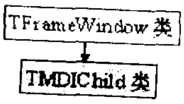


图 A－67　TMDIChild 类在 OWL 5.0 中的位置

Public 构造函数和析构函数：

```
TMDIChild(TMDIClient& parent,
        const char far * title = 0,
        TWindow * clientWnd = 0,
        bool shrinkToClient = false,
        TModule * module = 0);
TMDIChild(HWND hWnd,
        TModule * module = 0);
```

~TMDIChild();

Public 成员函数:

void Destroy(retVal = 0);

删除与 TMDIChild 相关的界面元素。

bool EnableWindow(bool enable);

重载 TWindow 的同名函数。

boolPreProcessMsg (MSG& msg);

重载 TWindow 的同名函数。

void ShowWindow(int cmdShow);

重载 TWindow 的同名函数。

Protected 成员函数:

LRESULT DefWindowProc (uint msg, WPARAM wParam, LPARAM lParam);

重载 TWindow 的同名函数。

void EvMDIActivate(HWND hWndActivated, HWND hWndDeactivated);

让一个客户窗口激活或非激活一个 MDI 子窗口，然后发出消息使子窗口激活或非激活。

bool EvNCActivate(bool active);

响应标题条或图标的改变事件。

void PerformCreate(int menuOrId);

创建与 MDI 子窗口相关的界面元素。

事件响应表：见表 A-75。

表 A-75

EV_ macros	成员函数
EV_WM_NCACTIVATE	EvNCActivate
EV_WM_MDIACTIVATE	EvMDIActivate

A.79　TDecoratedFrame 类

头文件：decframe.h

描述：TDecoratedFrame 类自动定位其客户窗口的位置并使客户窗口的大小等于客户区域的大小。你可以在窗口中增加一些“装饰品”，比如，工具条和状态条。你也可以创建一个没有标题条的 TDecoratedFrame 窗口。

TDecoratedFrame 类是可流类，它在 OWL 5.0 中的位置如图 A-68 所示。对于使用 OLE 功能的应用程序，应使用 TOleFrame 类。

类型定义：

```
enum TLocation(Top, Bottom, Left, Right);
```

Public 构造函数：

```
TDecoratedFrame(TWindow * parent,
                const char far * title,
                TWindow * clientWnd,
                bool trackMenuSelection = false,
                TModule * module = 0);
```

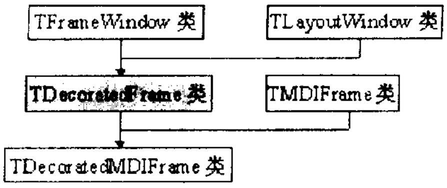


图 A－68　TDecoratedFrame 类在 OWL 5.0 中的位置

Public 成员函数：

void Insert(TWindow& decoration, TLocation = Top);

Insert 函数在客户窗口中放置修饰窗口。当包含多个修饰窗口时这个函数就更重要了，Insert 函数查看 decoration 的数据成员 Attr.Style；检查 WS_VISIBLE 标志以判别 decoration 在初始化时是显示还是隐藏；定位一个修饰窗口。Insert 还使用了 TLocation 数据类型，它描述了修饰窗口可以放置的四个位置。

bool PreProcessMsg(MSG& msg);

重载 TFrameWindow 中的同名函数。

TWindow * SetClientWindow(TWindow * clientWnd);

重载 TFrameWindow 中的同名函数。

Protected 数据成员：

uint MenuItemID;

菜单的 ID 值。

bool TrackMenuSelection;

是否显示菜单选择的帮助消息。

Protected 成员函数：

LRESULT EvCommand(uint Id, HWND hWndCtl, uint notifyCode);

自动显示和隐藏修饰物。

void EvCommandEnable(TCommandEnabler& ce);

处理与修饰物相关的菜单项的选择状态和非选择状态。

void EvEnterIdle(uint source, HWND hWndDlg);

响应窗口消息，告诉应用程序的子窗口一个对话框或菜单进入了空闲状态。

void EvMenuSelect(uint MenuItemId, uint flags, HMENU hMenu);

响应用户的选择菜单项的事件。

void EvSize(uint sizeType, TSize& size);

响应 WM_SIZE 消息，将它传给 TLayoutWindow。

void SetupWindow();

调用 TLayoutWindow::Layout 来调整修饰窗口的大小和位置。

事件响应表：见表 A－76。

表 A－76

EV_ macros	成员函数
EV_WM_ENTERIDLE	EvEnterIdle
EV_WM_MENUSELECT	EvMenuSelect
EV_WM_SIZE	EvSize

A. 80 TDecoratedMDIFrame 类

头文件：decmdifr.h

描述：TDecoratedMDIFrame 类从 TMDIFrame 类和 TDecoratedFrame 中派生出来。它作为一个 MDI 框架窗口可以包含修饰窗口。

TDecoratedMDIFrame 提供了用户工具条，你可以在框架窗口中插入一系列修饰物(比如，工具条和标尺)。当你需要不同的工具时，可以去掉原来的设置并重新插入其他的修饰窗口。在插入新的修饰窗口之前一定要在一定的位置上先去掉所有的旧修饰窗口。

TDecoratedMDIFrame 类是可流类，它在 OWL 5.0 中的位置如图 A－69 所示。

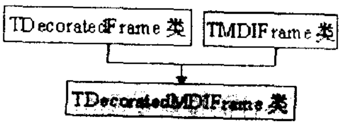


图 A－69 TDecoratedMDIFrame 类在 OWL 5.0 中的位置

Public 构造函数：

```
TDecoratedMDIFrame(const char far * title,
        TResId menuResId,
        TMDIClient &clientWnd = * new TMDIClient,
        bool trackMenuSelection = false,
        TModule * module = 0);
```

Protected 成员函数：

```
LRESULT DefWindowProc(uint message, WPARAM wParam, LPARAM lParam);
```

重载 TWindow 的同名函数。

A. 81 TGadgetWindow 类

头文件：gadgetwi.h

描述：TGadgetWindow 类是 TWindow 的派生类，用于建立一系列小器件(gadget)，使你能动态地安排工具条。你可以设置小器件的横向和纵向位置，字体，Left、right、top 和 bottom 边界，计算单位，修饰窗口大小等属性。

TGadgetWindow 是 TControlBar，TMessageBar，TToolBox，和 TStatusBar 等派生类的基类，它在 OWL 5.0 中的位置如图 A－70 所示。

Public 构造函数和析构函数：

```
TGadgetWindow(TWindow * parent = 0,
        TTileDirection direction = Horizontal,
        TFont * font = new TGadgetWindowFont,
```

TModule * module = 0);

~TGadgetWindow();

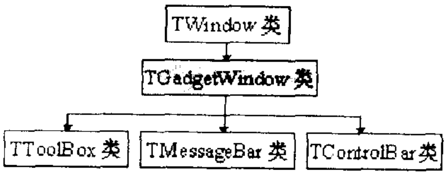


图 A-70　TGadgetWindow 类在 OWL 5.0 中的位置

定义类型：

enum THintMode{NoHints, PressHints, EnterHints};

Public 成员函数：

TGadget * FirstGadget() const;

返回小器件列表中的第一个小器件。

void GadgetChangedSize(TGadget& gadget);

当一个小器件改变了大小时调用 GadgetChangedSize，GradgetChangedSize 会调用 LayoutSession 来重新排列所有的小器件。

TGadget * GadgetFromPoint(TPoint& point);

按照所给的窗口坐标返回一个小器件的指针。

void GadgetReleaseCapture(TGadget& gadget);

释放鼠标的控制权，让其它窗口可以接收鼠标消息。

bool GadgetSetCapture(TGadget& gadget);

GadgetSetCapture 获得小器件窗口中所有的鼠标消息直到鼠标控制权被释放。

TGadget * GadgetWithId(int id) const;

返回一个指向具有特定 ID 值的小器件指针。

TTileDirection GetDirection() const;

返回小器件的排列方式。

TFont& GetFont();

返回使用的字体。

uint GetFontHeight() const;

返回字体的高度。

THintMode GetHintMode();

返回提示方式。

void IdleAction();

当没有事件需要处理时，调用此函数。

virtual void Insert(TGadget& gadget, TPlacement = After, TGadget * sibling = 0);

在一个兄弟小器件的前面或后面插入一个小器件，如果 sibling 的值为 0，那么新的小器件就插入在小器件列表的头或尾部；如果这个窗口已经建立了，在插入小器件后调用 LayoutSession 函数。

virtual void LayoutSession();

当增加和减少小器件或小器件边界的大小改变时，调用 LayoutSession 函数。LayoutSession 函数调用 TileGadgets 函数来以某种方式排列小器件。

TGadget * NextGadget(TGadget& gadget) const;

返回指定小器件的下一个小器件的指针，如果下一个小器件不存在则返回 0。

virtual TGadget * Remove(TGadget& gadget);

在小器件窗口中删除一个小器件，该小器件被去掉但是并不破坏。如果没有指定的小器件则返回 0；如果这个窗口已经建立了，在删除小器件后调用 LayoutSession 函数。

virtual void SetDirection(TTileDirection direction);

设置排列小器件的横向和纵向方式。在一个小器件窗口已经创建后，SetDirection 函数重新调整小器件

窗口的尺寸来适应小器件。

void SetHintCommand(int id);

模拟菜单项选择消息，这样 ObjectWindows 的 command 处理函数就可以显示菜单项的命令提示。

void SetHintMode(THintMode hintMode);

设置显示提示方式。缺省的显示方式是 PressHints(当小器件按下时显示提示)。

void SetMargins(TMargins& margins);

设置或改变小器件窗口的边界，调用 LayoutSession 函数。

void SetShrinkWrap(bool shrinkWrapWidth, bool shrinkWrapHeight);

设置数据成员的宽度和长度。缺省时，如果排列方向是 horizontal，则 ShrinkWrapWidth 是 false，ShrinkWrapHeight 是 true；当排列方向是 vertical 时，则 ShrinkWrapWidth 是 true，ShrinkWrapHeight 是 false。

Protected 数据成员：

TGadget * AtMouse;

鼠标位置上的最后一个小器件。

TBrush * BkgndBrush;

背景颜色的刷子。

TGadget * Capture;

指向拥有鼠标控制权的小器件；当没有小器件拥有鼠标控制权时，Capture 的值为 0。

TTileDirection Direction;

小器件排列方向，包括 horizontal(横向)和 vertical(纵向)。

bool DirtyLayout;

指明排列已经改变，小器件需要重新排列。使用 DirtyLayout 可避免小器件窗口创建时多余的排列。

TFont * Font;

指向字体的指针。

uint FontHeight;

指向小器件窗口的字体高度的指针。

TGadget * Gadgets;

指向小器件列表中的第一个小器件的指针。

THintMode HintMode;

提示方式。

TMargins Margins;

小器件窗口的边界值。

uint NumGadgets;

小器件窗口中的小器件数目。

bool ShrinkWrapHeight;

如果 ShrinkWrapHeight 是 true，则横向排列小器件。

bool ShrinkWrapWidth;

如果 ShrinkWrapWidth 是 true，则纵向排列小器件。

uint WideAsPossible;

所能排列小器件的数目。

Protected 数据成员：

bool Create();

重载 TWindow 类的同名函数。

void EvLButtonDown(uint modKeys, TPoint& point);

响应鼠标左按钮按下的消息，将鼠标消息传给所在的小器件。

void EvLButtonUp(uint modKeys, TPoint& point);

响应鼠标左按钮抬起的消息，将鼠标消息传给所在的小器件。

void EvMouseMove(uint modKeys, TPoint& point);

响应鼠标移动的消息，将鼠标消息传给所在的小器件。

void EvSize(uint sizeType, TSize& size);

调用 TWindow::EvSize 函数。如果 DirtyLayout 的值为 true 并且 WideAsPossible 大于 0，EvSize 设置 DirtyLayout 为 true 并调用 TileGadgets 来调节大小。

void EvSysColorChange();

当系统的颜色改变时调用 EvSysColorChange。

virtual void GetDesiredSize(TSize& size);

获得适当的小器件窗口大小。

virtual void GetInnerRect(TRect& rect);

计算小器件的区域。

void GetMargins(TMargins& margins, int& left, int& right, int& top, int& bottom);

返回上、下、左、右边界的尺寸。

int LayoutUnitsToPixels(int units);

将排列单位改变为像素点。

void Paint(TDC& dc, bool erase, TRect& rect);

将字体加到设备上下文中，调用 PaintGadgets 函数。

virtual void PaintGadgets(TDC& dc, bool erase, TRect& rect);

重新绘制窗口中的小器件。

virtual void PositionGadget(TGadget * previous, TGadget * next, TPoint& point);

调节小器件的位置。

virtual void TileGadgets();

以一定的方式重新排列小器件。

事件响应表：见表 A-77。

表 A-77

EV_ macros	成员函数
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_SIZE	EvSize
EV_WM_SYSCOLORCHANGE	EvSysColorChange

A.82　TToolBox 类

头文件：toolbox.h

描述：作为 TGadgetWindow 的派生类，TToolBox 以矩阵方式排列小器件。排列时，所有的行都有相同的宽度(与最宽的小器件一样宽)，所有的列都有相同的高度(与最高的小器件一样高)。你可以明确的说明你想要在工具箱中安排多少行多少列，也可以让工具箱自己计算所需要的行列数。如果你定义了 AS_MANY_AS_NEEDED 值，TToolBox 就计算在与之相对的尺寸中需要排列多少行和多少列。比如，如果有 20 个小器件需要排列，你要排成 4 行，那么矩阵的列就是 5。

TToolBox 类在 OWL 中的位置如图 A－71 所示。

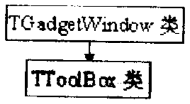


图 A－71　TTolBox 类在 OWL 5.0 中的位置

Public 构造函数：

```
TToolBox(TWindow * parent,
         int numColumns = 2,
         int numRows = AS_MANY_AS_NEEDED,
         TTileDirection direction = Horizontal,
         TModule * module = 0);
```

Public 成员函数：

void GetDesiredSize(TSize& size);

控制 TGadget 的 GetDesiredSize 函数，通过调用 GetMargins 函数获得边界来计算单元的大小。

void Insert(TGadget& gadget, TPlacement = After, TGadget * sibling = 0);

控制 TGadget 的 Insert 函数。

void LayoutSession();

当工具箱的边界或其中的小器件改变大小时调用此函数。LayoutSession 获得希望边界或小器件的大小并移动窗口调节尺寸。

virtual void SetDirection(TTileDirection direction);

设置小器件的排列方式是横向排列还是纵向排列。

Protected 数据成员：

int NumColumns;

工具箱中小器件排列的列数。

int NumRows;

工具箱中小器件排列的行数。

Protected 数据成员：

TRect TileGadgets;

以横向或纵向方式排列小器件时定义的空间。

A.83　TControlBar 类

头文件：controlb.h

描述：作为 TGadgetWindow 的派生类，TControlBar 完成一个控件条的功能。TControlBar 类可以安排一系列小按钮，它在 OWL 5.0 中的位置如图 A-72 所示。

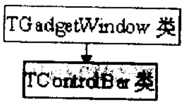


图 A-72　TControlBar 类在 OWL 5.0 中的位置

Public 构造函数：

```
TControlBar(TWindow * parent = 0,
            TTileDirection direction = Horizontal,
            TFont * font = new TGadgetWindowFont,
            TModule * module = 0);
```

Public 成员函数：

```
bool PreProcessMsg(MSG& msg);
```

预处理消息函数。

Protected 数据成员：

```
void PositionGadget(TGadget * previous, TGadget * next, TPoint& p);
```

获得边界风格，决定小器件的排列方向并安排小按钮的位置。

A.84　TMessageBar 类

头文件：messageb.h

描述：作为 TGadgetWindow 的派生类，TMessageBar 与小文本框一起构成一个信息条，这个信息条与窗口一样宽且没有边界。通常信息条在窗口的最下方，使用缺省的小器件窗口字体并在信息条的顶部画一条高亮线。TMessageBar 类在 OWL 5.0 中的位置如图 A-73 所示。

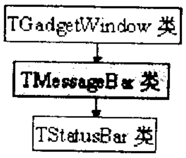


图 A-73　TMessageBar 类在 OWL 5.0 中的位置

Public 构造函数：

```
TMessageBar(TWindow * parent = 0,
            TFont * font = new TGadgetWindowFont,
            TModule * module = 0);
```

Public 成员函数：

```
virtual void SetHintText(const char * text);
```

设置或清除信息条中的菜单项提示文字。提示文字覆盖在所有的小器件上面，用来作为菜单项和控件条按钮的帮助信息。

```
void SetText(const char * text);
```

将菜单项的提示文字送到信息条的小文本框中。

Protected 数据成员：

```
bool HighlightLine;
```

表明是否绘制高亮线。

char * HintText;

保存当前正在显示的提示信息。

Protected 成员函数：

void GetDesiredSize(TSize& rect);

调用 TGadgetWindow 的 GetDesiredSize 函数来获得信息条的大小，然后，在需要绘制一条高亮线时调整信息条的大小。

void GetInnerRect(TRect& rect);

GetInnerRect 函数计算在信息条边界中的区域。

void PaintGadgets(TDC& dc, bool erase, TRect& rect);

调整信息条并绘制一条高亮线，如果所有的设置都完成了 PaintGadgets 就绘制提示信息，否则它就调用 TGadgetWindow::PaintGadgets 来重画所有的小器件。

A.85 TStatusBar 类

头文件： statusba.h

描述： TStatusBar 提供了状态条的功能，状态条中可以插入小器件、提示文字和键盘模式。状态条与信息条不同，它有多种显示方式。ObjectWindows 的状态条可以包括多个小文本框(在状态条的左边)，可具有不同的边界风格。你还可以保留一块区域作模式描述(显示应用程序当前的状态，包括键盘或其它状态，比如 CapsLock，NumLock，ScrollLock，Overtype，和 macro recording)。缺省时，ObjectWindows 使用这些虚键代码并显示表 A-78 所示的模式描述字符串。

表 A-78

模 式	虚键代码	文字显示
Extended		EXT
CapsLock	VK_CAPITAL	CAPS
NumLock	VK_NUMLOCK	NUM
ScrollLock	VK_SCROLL	SCRL
Overtype	VK_INSERT	OVR
Macro	recording	REC

TStatusBar 为你需要的模式描述创建小文本框并调节模式描述的位置。TSpacing 结构包含了描述的空间和排列。与其它的控件条一样，状态条与主窗口一起创建和删除，但这并不需要过程说明。下面的程序段说明了怎样创建一个状态条并将它插入到窗口的底部。

```
TStatusBar * sb = new TStatusBar(0, TGadget_Recessed,
    TStatusBar_CapsLock | TStatusBar_NumLock | TStatusBar_Overtype);
frame->Insert( * sb, TDecoratedFrame::Bottom);
MainWindow = frame;
```

TStatusBar 在 OWL 5.0 中的位置如图 A-74 所示。

定义类型：

enum TModeIndicator {ExtendSelection = 1,

```
        CapsLock = 1 << 1,
        NumLock = 1 << 2,
        ScrollLock = 1 << 3,
        Overtype = 1 << 4,
        RecordingMacro = 1 << 5};
```

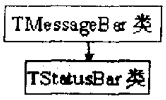


图 A-74　TstatusBar 类在 OWL 5.0 中的位置

Public 构造函数：

```
TStatusBar(TWindow * parent = 0,
        TGadget::TBorderStyle borderStyle = TGadget::Recessed,
        uint modeIndicators = 0,
        TFont * font = new TGadgetWindowFont,
        TModule * module = 0);
```

Public 成员函数：

bool GetModeIndicator(TModeIndicator i) const;

返回状态条的当前模式描述。

void Insert(TGadget& gadget, TPlacement = After, TGadget * sibling = 0);

在状态条中插入小器件(TGadget 和其派生类的对象)。缺省时，新的小器件放置在所有已存在小器件的后面并在状态条模式描述的左边。

TGadget * operator[](uint index);

获得 index 值表示的小器件，但不能访问模式描述。

void SetModeIndicator(TModeIndicator, bool state);

设置 TModeIndicator 为所给的小文本框设置状态条模式描述的开关。为了使模式描述显示在状态条上，你必须在窗口创建时设置描述。

void SetSpacing(TSpacing& spacing);

使用 TSpacing 值来设置模式描述之间的空间。通常，信息描述(最左边的小文本框)是左对齐的，其它的描述是右对齐的。

```
struct TSpacing
{
    TMargins::TUnits Units;
    int Value;
    TSpacing() {Units = TMargins::LayoutUnits; Value = 0;}
};
void ToggleModeIndicator(TModeIndicator);
```

Protected 数据成员：

TGadget::TBorderStyle BorderStyle;

保存状态条中的模式描述的边界风格，包括 None, Plain, Raised, Recessed, 或 Embossed。

uint McdeIndicators;

ModeIndicators 位字段描述了状态条创建了的模式描述种类。

uint ModeIndicatorState;

定义了状态条的模式，这些模式可以是 TModeIndicator 枚举中的值，包括，CapsLock, NumLock, ScrollLock, Overtype, RecordingMacro 和 ExtendSelection。

uint NumModeIndicators;

状态条中的模式描述数目(从1到5)。

TSpacing Spacing;

模式描述之间的距离。

Protected 数据成员:

bool IdleAction(long);

如果不只一个应用程序在运行,TStatusBar 就调用 IdleAction 函数而不是 PreProcessMsg 函数来检查 NumLock, CapsLock 和 ScrollLock 键的状态。

void PositionGadget(TGadget * previous, TGadget * next, TPoint& point);

根据已存在的小器件来决定新的小器件的相对位置。

bool PreProcessMsg(MSG& msg);

重载 TWindow::PreProcessMsg 来处理键盘消息并在 NumLock, CapsLock 或 ScrollLock 键按下时更新模式描述。

A.86　TGadget 类

头文件: gadget.h

描述: TGadget 类在 OWL 5.0中的位置如图 A-75所示,它是各种小器件类的基类,这些小器件类包括:TBitmapGadget、TButtonGadget、TControlGadget、TTextGadget、TSeparatorGadget。

TGadget 的界面对象属于一个小器件窗口,这个小器件窗口具有边框、边界和自己的坐标系统。你可以选择边框风格或者设置边框并重载成员函数 PaintBorder 来创建你自己的小器件边框。如果你改变了边框、边界或边框风格,小器件窗口的 GadgetChangedSize 数据成员就被调用。

虽然缺省时小器件自动适应所在的空间,但你也可以通过设置自己的 ShrinkWrapWidth 和 ShrinkWrapHeight 值来调节这个属性。

小器件窗口作为活动窗口可以接收鼠标事件。当小器件窗口接收到了消息后,会判断哪个小器件应该接收这个消息并直接调用成员函数来处理消息。

Public 数据成员:

bool Clip;

如果 Clip 的值是 false,不建立限制边框;如果 Clip 是 true,在绘制每一个小器件时都将其局限在边框中。

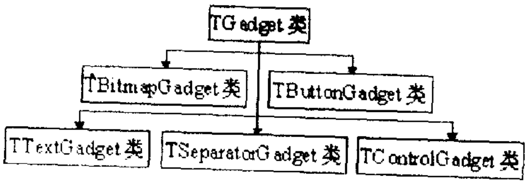


图 A-75　TGadget 类在 OWL 5.0中的位置

bool WideAsPossible;

WideAsPossible 表明是否使小器件的宽度在小器件窗口内尽可能宽,其初始设置是 false。

Public 枚举类型和结构:

struct Tborders

unsigned Left;

unsigned Right;

unsigned Top;

unsigned Bottom;

保存小器件的尺寸。

enum TBorderStyle{None, Plain, Raised, Recessed, Embossed};

小器件的边框风格。

struct Tmargins

小器件的边界信息。

Public 构造函数和析构函数：

TGadget(int id = 0, TBorderStyle = None);

virtual ~TGadget();

Public 成员函数：

virtual void CommandEnable();

这个函数使得小器件可以执行命令开关功能。

TBorders& GetBorders();

获得小器件的边框。

TBorderStyle GetBorderStyle();

获得小器件的边框风格。

TRect& GetBounds();

获得小器件的边界区域。

virtual void GetDesiredSize(TSize& size);

获得小器件的大小。

bool GetEnabled();

决定小器件是否可以获得键盘或鼠标的输入。

int GetId();

获得小器件的 ID 值。

TMargins& GetMargins();

获得小器件的边界尺寸。

void GetOuterSizes(int& left, int& right, int& top, int& bottom);

获得边界和边框内的尺寸(像素点)。

TGadget * NextGadget();

指向小器件列表中的下一个小器件的指针。

void SetBorders(TBorders& borders);

设置小器件的边框。

void SetBorderStyle(TBorderStyle);

设置小器件的边框风格。

virtual void SetBounds(TRect& rect);

通知小器件改变边界。

virtual void SetEnabled(bool);

设置小器件是否可以接收鼠标和键盘的输入。

void SetMargins(TMargins& margins);

设置边界。

void SetShrinkWrap(bool shrinkWrapWidth, bool shrinkWrapHeight);

设置 ShrinkWrapWidth 和 ShrinkWrapHeight 数据成员。

void SetSize(TSize& size);

改变小器件的大小，然后调用 TGadgetWindow::GadgetChangedSize 函数产生效果。

virtual void SysColorChange()；

当系统颜色改变并且小器件需要重画时调用此函数。

Protected 数据成员：

TRect Bounds；

小器件在小器件窗口中的位置坐标。

TBorders Borders；

```
struct Tborders
    unsigned Left;
    unsigned Right;
    unsigned Top;
    unsigned Bottom;
```

保存小器件的尺寸。

int Id；

小器件的 ID 值。

TMargins Margins；

```
struct TMargins
{
    enum TUnits {Pixels, LayoutUnits, BorderUnits};
    TUnits Units;
    int Left;
    int Right;
    int Top;
    int Bottom;
}
```

小器件的边界测量信息，供 TGadgetWindow 和 TGadget 使用。

bool ShrinkWrapHeight；

表明小器件以 shrink－wrapped 方式填充区域。

bool ShrinkWrapWidth；

表明小器件以 shrink－wrapped 方式填充区域。

bool TrackMouse；

初始化时为 false。当 TrackMouse 为 true 时，小器件获得鼠标的控制权。

TGadgetWindow * Window；

指向拥有小器件的小器件窗口。

Protected 数据成员：

void GetInnerRect(TRect& rect)；

计算包括边框在内的小器件区域。

virtual void Inserted()；

窗口中插入一个小器件时调用。

void Invalidate(bool erase = true)；

使小器件区域失效。

void InvalidateRect(const TRect& rect, bool erase = true);

使窗口中与小器件有关的区域失效。

virtual void LButtonDown(uint modKeys, TPoint& point);

如果设置了 TrackMouse，就响应鼠标。point 是鼠标在小器件坐标系统中的位置。

virtual void LButtonUp(uint modKeys, TPoint& point);

如果设置了 TrackMouse，就响应鼠标。point 是鼠标在小器件坐标系统中的位置。

virtual void MouseEnter(uint modKeys, TPoint& point);

鼠标进入小器件时调用。

virtual void MouseLeave(uint modKeys, TPoint& point);

鼠标离开小器件时调用。

virtual void MouseMove(uint modKeys, TPoint& point);

如果设置了 TrackMouse，就响应鼠标拖动。point 是鼠标在小器件坐标系统中的位置。

virtual void Paint(TDC&);

调用 PaintBorder 在所给的设备上下文中绘制。

virtual void PaintBorder(TDC& dc);

绘制小器件的边框。

virtual bool PtIn(TPoint& point);

如果所给的一点在小器件的边界内，返回 true;否则返回 false。

virtual void Removed();

当小器件从一个窗口删除的时候调用。

void Update();

在必要时重新绘制小器件。

A.87　TBitmapGadget 类

头文件：bitmapga.h

描述：作为 TGadget 类的派生类，TBitmapGadget 是一个可以显示一系列小位图的小器件，它在 OWL 5.0中位置如图 A-76所示。

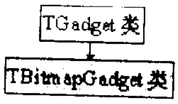


图 A-76　TBitmapGadget 类在 OWL 5.0中的位置

Public 构造函数和析构函数：

```
TBitmapGadget(TResId bmpResId,
              int id,
              TBorderStyle borderStyle,
              int numImages,
              int startImage);
~TBitmapGadget();
```

Public 成员函数：

int SelectImage(int imageNum, bool immediate);

选择当前的位图，如果当前位图改变了就重画客户区域。

void SysColorChange();

当系统的颜色改变时，SysColorChange 函数被小器件窗口的 EvSysColorChange 函数调用，使小位图得以重画或重建。

Protected 数据成员：

void GetDesiredSize(TSize& size);

调用 TGadget::GetDesiredSize 函数决定小位图的大小。

void Paint(TDC& dc);

绘制小位图的边框和内容。

void SetBounds(TRect& r);

调用 TGadget::SetBounds 函数传输小位图的尺寸。

A.88　TButtonGadget 类

头文件： buttonga.h

描述： TButtonGadget 类在 OWL 5.0中的位置如图 A-77所示。作为 TGadget 类的派生类，TButtonGadgets 代表用户可以开关的小按钮，你可以设置这些小按钮的属性参数，比如，颜色、风格和形状等。

通常，小按钮分为命令按钮和属性按钮两类。属性按钮包括了 radio 按钮(排他的)和检查框(不排他的)，数据成员 TType 枚举了按钮的种类。

TButtonGadget 类包括了一系列可改变按钮的风格的函数。其中，SetAntialiasEdges 函数用来设置开关按钮，SetNotchCorners 函数用来控制角的样式，SetShadowStyle 函数用来改变按钮的阴影风格。

TButtonGadget 对象以下面的方式响应鼠标事件：当鼠标按键按下时，小按钮按下；当鼠标按键释放时，小按钮释放。当鼠标按钮的状态处于“up”状态时，命令才传入。当鼠标按钮被按下时，TButtonGadget 对象获得了鼠标的控制权，接收鼠标在当前窗口的全部消息。当鼠标按钮抬起时，小按钮释放鼠标控制权给当前窗口。

定义类型：

enum TShadowStyle{SingleShadow = 1, DoubleShadow = 2};

枚举按钮的阴影风格。

enum TState{Up, Down, Indeterminate};

枚举小按钮的三种状态。

enum TType{Command, Exclusive, NonExclusive};

枚举小按钮的类型：命令小按钮、排他小按钮和不排他小按钮。一组排他小按钮中在一个时刻只能有一个小按钮被设置为按下状态。

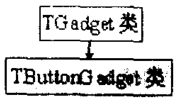


图 A-77　TButtonGadget 类在 OWL 5.0中的位置

Public 构造函数和析构函数：

```
TButtonGadget(TResId bmpResId,
          int id,
          TType type = Command,
          bool enabled = false,
          TState state = Up,
          bool repeat = false);
~TButtonGadget();
```

Public 成员函数：

void CommandEnable();

开启小按钮的接收消息功能，调用 SendMessage 函数发出一个 WM_COMMAND_ENABLE 消息给小器件窗口的父窗口并为这个小按钮传输一个 TCommandEnable：EvCommandEnable 消息参数。

TState GetButtonState()；

返回按钮的状态，如果为0，说明按钮是抬起状态；如果是1，说明按钮是按下状态；如果是2，说明按钮处在不确定状态。

TType GetButtonType()；

返回按钮的类型，如果按钮是命令按钮则是1，如果是排他按钮则是2，如果是不排他按钮则是3。

void SetAntialiasEdges(bool anti = true)；

设置数据成员 AntialiasEdges 的开关，缺省时为开。

void SetButtonState(TState)；

设置小按钮的状态，如果状态改变了并且按钮是排他的，则在在按下状态时检查其他排他按钮，并设置相应状态。

void SetNotchCorners(bool notchCorners = true)；

缺省时，SetNotchCorners 表明按钮的切割角方式。

voidSetShadowStyle(TShadowStyle)；

设置按钮的阴影风格参数，缺省时是 DoubleShadow；设置按钮的左边和上边为2，右边和下边为 ShadowStyle + 1。

void SysColorChange()；

SysColorChange 函数响应一个 EvSysColorChange 消息。

Protected 数据成员：

bool AntialiasEdges；

如果 AntialiasEdges 功能开启，其值为 true。

TPoint BitmapOrigin；

指向小按钮所使用的位图的坐标。

TCelArray * CelArray；

指向小按钮使用的特定 Cel 数组。

bool NotchCorners；

初始化设置为1，NotchCorners 为1说明小按钮有切割角，否则没有切割角。

bool Pressed；

如果按钮被放开 Pressed 的值为1，如果没有放开按钮则值为0，初始化设置为1。

bool Repeat；

Repeat 存储了键盘事件的重复次数，初始化设置为1。

TResId ResId；

存储了小按钮的位图 ID 值。

TShadowStyle ShadowStyle；

小按钮的阴影风格：1是单阴影，2为双阴影。

TState State；

小按钮的状态信息：按下、抬起或不确定状态。

TType Type；

小按钮的类型：命令、排他和不排他按钮。

Protected 成员函数：

virtual void Activate(TPoint& p);

当鼠标键的状态是"up"时调用此函数。Activate 函数设置 Pressed 参数为 false，改变属性按钮的状态。

virtual void BeginPressed(TPoint& p);

当鼠标按键按下时，Beginpressed 函数设置 Pressed 为 true。

virtual void BuildCelArray();

根据资源位图创建 cel 数组，当新的 cel 数组创建时已存在的 cel 数组就被删除。

virtual void CancelPressed(TPoint& p);

当鼠标按键释放时，CancelPressed 设置 Pressed 为 false。

void GetDesiredSize(TSize& size);

保存小按钮的宽度和高度(以像素点为单位)，调用 TGadget 类的 GetDesiredSize 函数来计算区域之间的关系。

virtual TDib * GetGlyphDib();

提供 glyphdib。你可以重载这个函数来获得不同的 dib 或者改变 dib 的属性。

void Invalidate();

如果按钮被按下或是状态改变了，Invalidate 函数就使改变的部分变为失效，调用 TGadget::InvalidateRect 函数重画按钮。

void LButtonDown(uint modKeys, TPoint& p);

重载 TGadget 类的成员函数，调用 BeginPressed 函数响应鼠标左键按下的事件。

void LButtonUp(uint modKeys, TPoint& p);

重载 TGadget 类的成员函数，调用 Activate 函数响应鼠标左键释放的事件。

void MouseEnter(uint modKeys, TPoint& p);

当鼠标进入小按钮区域时调用此函数。

void MouseLeave(uint modKeys, TPoint& p);

当鼠标离开小按钮区域时调用此函数。

void MouseMove(uint modKeys, TPoint& p);

调用 TGadget::MouseMove 函数响应鼠标拖动的事件。

void Paint(TDC& dc);

调用 TGadget::PaintBorder 函数在指定的设备上下文中绘制。

virtual void ReleaseGlyphDib(TDib * glyph);

释放 glyph dib。

void SetBounds(TRect& r);

获得位图的大小，调用 TGadget::SetBounds 函数来设置边框。

A.89 TTextGadget 类

头文件：textgadg.h

描述：作为 TGadget 类的派生类，TTextGadget 类提供小文本。在构造小文本对象时，你必须明确地知道你要显示多少字符，文本以何种方式对齐。小文本的内边框是由其中所能显示的最多字符数决定的。TTextGadget 在 OWL 5.0中的位置如图 A-78所示。

Public 构造函数和析构函数：

```
TTextGadget(int id = 0,
            TBorderStyle = Recessed,
```

```
TAlign = Left,
uint numChars = 10,
const char * text = 0); # =~TTextGadget();
```

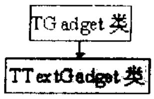


图 A-78 TTextGadget 类在 OWL 5.0中的位置

Public 成员函数：

char * GetText();

返回小文本中的字符串。

void SetText(const char * text);

如果 Text 中的字符串与新的字符串 text 不同，SetText 函数就删除 Text 中的字符串并将 TextLen 设置为新的字符串的长度。如果 text 为空，那么就设置 Text 和 TextLen 的值为0，然后调用 Invalidate 函数使这个区域失效。

Protected 数据成员：

TAlign Align;

文本的对齐方式：左、居中和右对齐。

uint NumChars;

小文本中的字符数。

char * Text;

小文本中的字符串。

uint TextLen;

小文本中的字符串长度。

Protected 成员函数：

void GetDesiredSize(TSize &size);

返回小文本的区域。

void Invalidate();

将小文本区域设置为失效。

void Paint(TDC& dc);

调用 TGadget::PaintBorder 函数在指定的设备上下文中绘制。

定义类型：

enum TTileDirection{Horizontal, Vertical};

枚举小文本允许的两种排列方向。

enum TAlign {Left, Center, Right};

枚举小文本中文本的对齐方式：左、居中和居右对齐。

A.90 TSeparatorGadget 类

头文件： gadget.h

描述： TSeparatorGadget 类提供了小器件之间的间隔，此间隔的大小可以根据实际情况调整。TSeparatorGadget 类在 OWL 5.0中的位置如图 A-79所示。

Public 成员函数：

TSeparatorGadget(int size = 6);

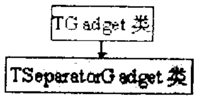


图 A-79　TSeparatorGadget 类在 OWL 5.0中的位置

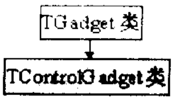


图 A-80　TControlGadget 类在 OWL 5.0中的位置

A.91　TControlGadget 类

头文件：controlg.h

描述：TControlGadget 类作用就是使得小器件窗口中可以包含 TControl 对象，比如可以将编辑框、按钮等加入到小器件窗口中。TControlGadget 类在 OWL 5.0中的位置如图 A-80所示。

Public 构造函数和析构函数：

TControlGadget(TWindow& control, TBorderStyle = None);

~TControlGadget();

Protected 数据成员：

TWindow * Control;

被小控件管理的控件对象。

Protected 成员函数：

void GetDesiredSize(TSize& size);

调用 TGadget::GetDesiredSize 函数获得小控件所需的区域大小。

void GetInnerRect(TRect&);

计算除了边框和边界以外的小控件区域。

void Inserted();

当小控件插入到父窗口中时调用此函数，以当前大小和位置显示窗口。

void Invalidate(bool erase = true);

使小控件区域失效。

void InvalidateRect(const TRect&, bool erase = true);

使小控件中指定的区域失效。

void Removed();

当父窗口删除小控件时调用此函数。

void SetBounds(TRect& rect);

调用 TGadget::SetBounds 函数设置边框。

void Update();

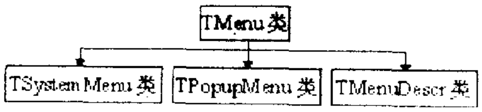
通过发出 WM_PAINT 消息更新窗口客户区域。

A.92 TMenu类

头文件：menu.h

描述：TMenu类封装了窗口的菜单操作。你可以使用TMenu的成员函数来构造、改变、查询和创建一个菜单对象，你还可以使用TMenu在你的菜单中增加位图，设置选择状态等。

TMenu类在OWL 5.0中的位置如图A-81所示。



图A-81　TMenu类在OWL 5.0中的位置

Public 构造函数和析构函数：

TMenu(TAutoDelete autoDelete = AutoDelete);

创建一个空的菜单并设置AutoDelete的值。autoDelete值缺省时，在对象删除时的同自动删除菜单。

TMenu(HWND hWnd, TAutoDelete autoDelete = NoAutoDelete);

创建一个已存在菜单的拷贝并设置autoDelete的值。autoDelete值缺省时，在对象删除时不会自动删除菜单。

TMenu(const TMenu& original, TAutoDelete autoDelete = AutoDelete);

创建一个菜单对象，作为窗口的当前菜单并设置autoDelete的值。autoDelete值缺省时，在对象删除时不会自动删除菜单。

TMenu(HMENU handle, TAutoDelete autoDelete = NoAutoDelete);

从一个以经装载的菜单创建一个对象并设置autoDelete的值。autoDelete值缺省时，在对象删除时不会自动删除菜单。

TMenu(const void far * menuTemplate);

从内存中的样板菜单创建一个菜单对象。

TMenu(HINSTANCE instance, TResId resId);

从一个所给的资源创建一个菜单对象。

virtual ~TMenu();

Public 成员函数：

bool AppendMenu(uint flags, uint newItem, const TBitmap& newBmp);

在窗口的尾部增加一个位图菜单项。

bool AppendMenu(uint flags, uint newItem = -1, const char far * newItem = 0);

在窗口的尾部增加一个位图菜单项。

bool CheckMenuItem(uint item, uint check);

设置一个菜单项的选择状态。

void CheckValid(uint resId = IDS_MENUFAILURE);

检查菜单是否有效。

bool DeleteMenu(uint item, uint flags);

从菜单中删除一个菜单项。

virtual void DrawItem(DRAWITEMSTRUCT far& drawItem);

响应一个绘制消息。

bool EnableMenuItem(uint item, uint enable);

设置菜单项功能开启、关闭和变为灰色，设置成功则返回true。如果菜单项功能开启，用户就可以选

择这个菜单项；如果菜单项功能关闭，菜单项就不会被显示；如果菜单项变为灰色，用户可以看到这个菜单项，但不能选择它。

virtual HMENU GetHandle(uint item, uint enable);

返回菜单的句柄。

static bool GetMenuCheckMarkDimensions(TSize& size);

获得位图的大小，这个位图显示在菜单项的前面表示菜单项的状态为选择状态。

uint GetMenuItemCount() const;

获得菜单中的菜单项的数目。

uint GetMenuItemID(int posItem) const;

根据所给的位置确定菜单项的 ID 值。

uint GetMenuState(uint item, uint flags) const;

返回所给菜单项的状态。如果菜单不是弹出菜单，则返回值的取值组合和意义如表 A-79所示，如果菜单项不存在就返回-1。

表 A-79

取　值	意　　义
MF_BITMAP	菜单项是一个位图
MF_CHECKED	菜单项被选择了(菜单是弹出菜单)
MF_DISABLED	菜单项的功能被关闭了
MF_ENABLED	菜单项的功能开启了
MF_GRAYED	菜单项的变灰
MF_MENUBARBREAK	与 MF_MENUBREAK 显示，只是弹出菜单的分割线是垂直的
MF_MENUBREAK	静态菜单项位于一个新的行，弹出菜单项位于一个新的列，并且没有分割线
MF_SEPARATOR	水平分割线
MF_UNCHECKED	菜单项的状态状态为非选择，即菜单项前面没有

uint GetMenuString(uint item, char * str, int count, uint flags) const;

获得菜单项的字符串。

HMENU GetSubMenu(int posItem) const;

获得子菜单的句柄。

bool InsertMenu(uint item, uint flags, uint newItem, const TBitmap& newBmp);

在所给的菜单项后面增加一个位图菜单项。

bool InsertMenu(uint item, uint flags, uint newItem = -1, const char far * newItem = 0);

在所给的菜单项后面增加一个菜单项。

bool IsOK() const;

如果菜单的句柄有效就返回 true。

virtual void MeasureItem(MEASUREITEMSTRUCT far& measureItem);

被自绘控件用来存储菜单的尺寸。

bool ModifyMenu(uint item, uint flags, uint newItem, const TBitmap& newBmp);

将菜单项改变为所给的位图菜单项。

bool ModifyMenu(uint item, uint flags, uint newItem = −1, const char far * newItem = 0);

将菜单项改变为所给的字符串菜单项。

operator HMENU()const;

返回菜单的句柄。

TMenu& operator =(const TMenu& original);

返回菜单对象。

operator uint()const;

返回菜单的句柄。

bool RemoveMenu(uint item, uint flags);

从菜单中删除菜单项，但不能删除一个子菜单。

```
bool SetMenuItemBitmaps(uint item,
            uint flags,
            const TBitmap * bmpUnchecked=0,
            const TBitmap * bmpChecked=0);
```

定义标志菜单项的选择状态的位图。

Protected 数据成员：

HMENU Handle;

菜单的句柄。

bool ShouldDelete;

表明在对象删除时是否删除菜单。

Protected 成员函数：

static void DeepCopy(TMenu& dest, const TMenu& source, int offset = 0, int count = −1);

完整拷贝一个菜单(不是只拷贝菜单对象的指针)。

static void DeepCopy(TMenu& dst, int dstOff, const TMenu& src, int srcOff = 0, int count = −1);

完整拷贝一个菜单(不是只拷贝菜单对象的指针)。

A.93　TSystemMenu 类

头文件： menu.h

描述： 作为 TMenu 类的派生类，TSystemMenu 类创建一个系统菜单对象，他在 OWL 5.0中的位置如图 A-82所示。

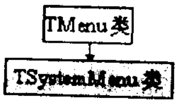


图 A-82　TSystemMenu 类在 OWL 5.0中的位置

Public 构造函数：

TSystemMenu(HWND wnd, bool revert = false);

A.94　TPopupMenu 类

头文件：menu.h

描述：作为 TMenu 类的派生类，TPopupMenu 类创建一个空的弹出菜单，可以将此弹出菜单增加到窗口菜单和弹出菜单中。

TPopMenu 类在 OWL 5.0中的位置如图 A-83所示。

Public 构造函数：

TPopupMenu(TAutoDelete autoDelete = AutoDelete);

TPopupMenu(HMENU handle, TAutoDelete autoDelete = NoAutoDelete);

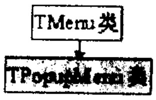


图 A-83　TPopupMenu 类在 OWL 5.0中的位置

Public 成员函数：

bool TrackPopupMenu(uint flags, int x, int y, int rsvd, HWND wnd, TRect * rect = 0);

在指定的位置创建弹出菜单。

bool TrackPopupMenu(uint flags, TPoint& point, int rsvd, HWND wnd, TRect * rect = 0);

在指定的位置创建弹出菜单。

A.95　TMenuDescr 类

头文件：menu.h

描述：作为 TMenu 类的派生类，TMenuDescr 类描述了菜单条和及其功能组。TMenuDescr 提供了一系列方便的办法来管理菜单项，你可将菜单项分组，在已存在的菜单中增加新的组。TMenuDescr 使用资源 ID 值来标志菜单，使用一系列数字来标志每一个菜单组中弹出菜单的值。联合类型 TGroup 包括了六个值：File、Edit、Container、Object、Window 和 Help，这是菜单条中最基本的六个菜单组。

TMenuDescr 类在 OWL 5.0中的位置如图 A-84所示。

Public 构造函数和析构函数：

TMenuDescr();

TMenuDescr(const TMenuDescr& original);

TMenuDescr(TResId id, TModule * module = ::Module);

TMenuDescr(TResId id,
        int fg, int eg, int cg, int og, int wg, int hg,
        TModule * module = ::Module);

TMenuDescr(HMENU hMenu,
        int fg, int eg, int cg, int og, int wg, int hg,
        TModule * module = ::Module);

~TMenuDescr();

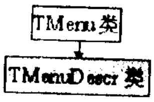


图 A-84　TMenuDescr 类在 OWL 5.0中的位置

类型定义：

enum TGroup{ FileGroup, EditGroup, ContainerGroup, ObjectGroup,

WindowGroup, HelpGroup, NumGroups};

Public 成员函数:

HMENU GetHandle() const ;

获得菜单的句柄。

TModule * GetModule() const;

获得模块指针。

void SetModule(TModule * module);

设置缺省的模块。

TResId GetId() const;

获得菜单的资源 ID 值。

int GetGroupCount(int group) const;

返回所给菜单组中的菜单数目。

void ClearServerGroupCount();

清除一些菜单组。

void ClearContainerGroupCount();

清除一些菜单组。

bool Merge(const TMenuDescr& sourceMenuDescr);

将其他菜单融合到此菜单中。

bool Merge(const TMenuDescr& sourceMenuDescr, TMenu& destMenu);

将其他菜单与此菜单融合到一个空菜单中。

Protected 数据成员:

TResId Id;

菜单的资源 ID 值。

int GroupCount[NumGroups];

菜单中的组的标志。

TModule * Module;

模块指针。

Protected 成员函数:

bool ExtractGroups();

从已装载的菜单中提取菜单组的数目。

附录B

OWL中的事件响应宏定义

B.1　事件响应类

在ObjectWindows应用程序中用事件响应表来处理事件，使用事件响应表时下面几个步骤是必需的：
声明事件响应表；
定义事件响应表；
定义事件响应表入口；
描述和定义响应函数。

B.2　声明事件响应表

因为事件响应表是ObjectWindows类中的一个成员，所以象其它成员一样，你需要在类声明中声明这个成员。ObjectWindows程序使用DECLARE_RESPONSE_TABLE宏代替实际的响应描述，比较简洁。

DECLARE_RESPONSE_TABLE宏只有一个参数，它就是响应这个事件的类名称。在类声明的最后声明这个宏，并且不必关心访问级别(比如：public、protected或private)。比如TNewWindow类的声明如下：

```
class TNewWindow ; public TWindow
{
……
    DECLARE_RESPONSE_TABLE(TNewWindow);
};
```

注意：事件响应表最好是在类声明的最后。

B.3　定义事件响应表

与其他的成员一样，事件响应表需要定义，而且需要在类的外部定义。ObjectWindows中使用DEFINE_RESPONSE_TABLEX宏开始定义，其中X表示在事件响应表的参数中排列的父类的数目(一般，除了当前的类外，还有它的一些父类也需要作为宏的参数)，因而，在宏定义DEFINE_RESPONSE_TABLEX中的参数个数就是X+1；使用END_RESPONSE_TABLE宏结束事件响应表的定义。例如：

```
DEFINE_RESPONSE_TABLE1(TNewWindow, TWindow)
  EV_WM_LBUTTONDOWN,
  EV_WM_LBUTTONUP,
  EV_WM_MOUSEMOVE,
  EV_WM_RBUTTONDOWN,
```

END_RESPONSE_TABLE;

在每一个响应声明后面跟逗号，END_RESPONSE_TABLE 宏后面则跟分号。

事件响应表将窗口事件和成员函数对应起来，使得这个成员函数可以响应这个事件。当窗口接到一个消息，它就检查在事件响应表中是否有对应的入口，如果有，就执行函数响应事件；如果没有，就将消息传给父窗口（如果是应用程序主窗口，就传给应用程序类）。应用程序对象也可以定义事件响应表。如果程序中没有这个消息的响应函数，它就被 ObjectWindows 缺省事件处理函数处理。

B.4　事件响应的宏

ObjectWindows 定义了许多事件响应入口的宏，主要包括：

命令消息宏，响应诸如菜单项命令等；

标准的窗口消息宏；

注册事件响应宏；

主窗口 ID 的广播事件响应宏；

控件广播事件响应宏；

文档管理消息响应宏；

VBX 控件响应。

下面以表格形式列出了这些宏及其参数。

按钮控件广播消息

表 B-1 所列的宏响应 BN_xxxx 消息。

表 B-1

宏	宏参数	响应函数声明
EV_BN_CLICKED	ID, UserName	void UserName()
EV_BN_DISABLE	ID, UserName	void UserName()
EV_BN_DOUBLECLICKED	D, UserName	void UserName()
EV_BN_HILITE	ID, UserName	void UserName()
EV_BN_PAINT	ID, UserName	void UserName()
EV_BN_UNHILITE	ID, UserName	void UserName()

子窗口 ID 广播消息

表 B-2 所列的宏响应子窗口发给父窗口的消息。

表 B-2

宏	宏参数	响应函数声明
EV_CHILD_NOTIFY_AND_CODE	ID, Code, UserName	void UserName(WPARAM)
EV_CHILD_NOTIFY_ALL_CODES	ID, UserName	void UserName(UINT)
EV_NOTIFY_AT_CHILD	Code, UserName	void UserName
EV_CHILD_NOTIFY	ID, Code, UserName	void UserName

组合框控件广播消息

表 B－3 所列的宏响应 CBN_xxxx 消息。

表 B－3

宏	宏参数	响应函数声明
EV_CBN_CLOSEUP	ID，UserName	void UserName()
EV_CBN_DBLCLK	ID，UserName	void UserName()
EV_CBN_DROPDOWN	ID，UserName	void UserName()
EV_CBN_EDITCHANGE	ID，UserName	void UserName()
EV_CBN_EDITUPDATE	ID，UserName	void UserName()
EV_CBN_ERRSPACE	ID，UserName	void UserName()
EV_CBN_KILLFOCUS	ID，UserName	void UserName()
EV_CBN_SELCHANGE	ID，UserName	void UserName()
EV_CBN_SELENDCANCEL	ID，UserName	void UserName()
EV_CBN_SELENDOK	ID，UserName	void UserName()
EV_CBN_SETFOCUS	ID，UserName	void UserName()

命令消息

表 B－4 所列的宏响应 WM_COMMAND 消息。

表 B－4

宏	宏参数	响应函数声明
EV_COMMAND	CMDID，UserName	void UserName()
EV_COMMAND_AND_ID	CMDID，UserName	void UserName(WPARAM)
EV_COMMAND_ENABLE	CMDID，UserName	void UserName(TCommandEnabler&)

文档管理消息

表 B－5 所列的宏响应文档管理器的消息。

表 B－5

宏	宏参数	响应函数声明
EV_OWLDOCUMENT	ID，UserName	void UserName(TDocument& document)
EV_OWLNOTIFY	ID，UserName	bool UserName(LPARAM&)
EV_OWLVIEW	ID，UserName	void UserName(TView& view)

文档观察器消息

表 B－6 所列的宏响应文档观察器的消息。

表 B-6

宏	响应函数声明
EV_VN_COMMIT	bool VnCommit(bool force)
EV_VN_DOCCLOSED	bool VnDocClosed(int openMode)
EV_VN_DOCOPENED	bool VnDocOpened(int openMode)
EV_VN_ISDIRTY	bool VnIsDirty()
EV_VN_ISWINDOW	bool VnIsWindow(HWND hWnd)
EV_VN_VIEWCLOSED	bool VnViewClosed(TView * view)
EV_VN_VIEWOPENED	bool VnViewOpened(TView * view)
EV_VN_REVERT	bool VnRevert(bool clear)

编辑框控件广播消息

表 B-7 所列的宏响应编辑框控件的 EN_xxxx 消息。

表 B-7

宏	宏参数	响应函数声明
EV_EN_CHANGE	ID, UserName	void UserName()
EV_EN_ERRSPACE	ID, UserName	void UserName()
EV_EN_HSCROLL	ID, UserName	void UserName()
EV_EN_KILLFOCUS	ID, UserName	void UserName()
EV_EN_MAXTEXT	ID, UserName	void UserName()
EV_EN_SETFOCUS	ID, UserName	void UserName()
EV_EN_UPDATE	ID, UserName	void UserName()
EV_EN_VSCROLL	ID, UserName	void UserName()

列表框控件广播消息

表 B-8 所列的宏处理列表框消息 LBN_xxxx。

表 B-8

宏	宏参数	响应函数声明
EV_LBN_DBLCLK	ID, UserName	void UserName()
EV_LBN_ERRSPACE	ID, UserName	void UserName()
EV_LBN_KILLFOCUS	ID, UserName	void UserName()
EV_LBN_SELCANCEL	ID, UserName	void UserName()
EV_LBN_SELCHANGE	ID, UserName	void UserName()
EV_LBN_SETFOCUS	ID, UserName	void UserName()

ObjectComponents 消息

表 B－9 所列的宏处理 ObjectComponents 消息。

表 B－9

宏	响应函数声明
EV_OC_APPBORDERSPACEREQ	bool EvOcAppBorderSpaceReq(TRect far * rect)
EV_OC_APPBORDERSPACESET	bool EvOcAppBorderSpaceSet(TRect far * rect)
EV_OC_APPDIALOGHELP	void EvOcAppDialogHelp(TOcDialogHelp far& dh)
EV_OC_APPFRAMERECT	bool EvOcAppFrameRect(TRect far * rect)
EV_OC_APPINSMENUS	bool EvOcAppInsMenus(TOcMenuDescr far&)
EV_OC_APPMENUS	bool EvOcAppMenus(TOcMenuDescr far& md)
EV_OC_APPPROCESSMSG	bool EvOcAppProcessMsg(MSG far * msg)
EV_OC_APPRESTOREUI	void EvOcAppRestoreUI()
EV_OC_APPSHUTDOWN	void EvOcAppShutDown()
EV_OC_APPSTATUSTEXT	void EvOcAppStatusText(const char far * rect)
EV_OC_VIEWATTACHWINDOW	bool EvOcViewAttachWindow()
EV_OC_VIEWBORDERSPACEREQ	bool EvOcViewBorderSpaceReq(TRect far * rect)
EV_OC_VIEWBORDERSPACESET	bool EvOcViewBorderSpaceSet(TRect far& rect)
EV_OC_BREAKLINK	bool EvOcViewBreakLink(TOcLinkView& view)
EV_OC_VIEWCLIPDATA	HANDLE EvOcViewClipData(TOcFormat far& format)
EV_OC_VIEWCLOSE	bool EvOcViewClose()
EV_OC_VIEWDRAG	bool EvOcViewDrag(TOcDragDrop far& ddInfo)
EV_OC_VIEWDROP	bool EvOcViewDrop(TOcDragDrop far& ddInfo)
EV_OC_VIEWGETPALETTE	bool EvOcViewGetPalette(LOGPALETTE far * far *)
EV_OC_VIEWGETSCALE	bool EvOcViewGetScale(TOcScaleFactor&)
EV_OC_VIEWGETSITERECT	bool EvOcViewGetSiteRect(TRect far * rect)
EV_OC_VIEWINSMENUS	bool EvOcViewInsMenus(TOcMenuDescr far&)
EV_OC_VIEWLOADPART	bool EvOcViewLoadPart(TOcSaveLoad far& ocLoad)
EV_OC_VIEWOPENDOC	bool EvOcViewOpenDoc(const char far * path)
EV_OC_VIEWPAINT	bool EvOcViewPaint(TOcViewPaint far& vp)
EV_OC_VIEWPARTACTIVATE	bool EvOcViewPartActivate(TOcPart& ocPart)
EV_OC_VIEWPARTINVALID	bool EvOcViewPartInvalid(TOcPart far& part)
EV_OC_VIEWPARTSIZE	bool EvOcViewPartSize(TRect far * size)
EV_OC_VIEWSAVEPART	bool EvOcViewSavePart(TOcSaveLoad far& ocSave)
EV_OC_VIEWSCROLL	bool EvOcViewScroll (TOcScrollDir scrollDir)
EV_OC_SETLINK	bool EvOcViewSetLink(TOcLinkView& view)
EV_OC_VIEWSETSCALE	bool EvOcViewSetScale(TOcScaleFactor&)
EV_OC_VIEWSETSITERECT	bool EvOcViewSetSiteRect(TRect far * rect)
EV_OC_VIEWSHOWTOOLS	bool EvOcViewShowTools(TOcToolBarInfo far& tbi)
EV_OC_VIEWTITLE	const char far * EvOcViewTitle()

滚动条控件广播消息

表 B－10 所列的宏响应滚动条消息 SB_xxxx。

表 B－10

宏	宏参数	响应函数声明
EV_SB_BEGINTRACK	ID, UserName	void UserName()
EV_SB_BOTTOM	ID, UserName	void UserName()
EV_SB_ENDSCROLL	ID, UserName	void UserName()
EV_SB_LINEDOWN	ID, UserName	void UserName()
EV_SB_LINEUP	ID, UserName	void UserName()
EV_SB_PAGEDOWN	ID UserName	void UserName()
EV_SB_PAGEUP	ID, UserName	void UserName()
EV_SB_THUMBPOSITION	ID, UserName	void UserName()
EV_SB_TOP	ID, UserName	void UserName()

标准窗口消息

表 B－11 所列的宏响应标准窗口消息。

表 B－11

宏	响应函数声明
EV_WM_ACTIVATE	void EvActivate(uint active, bool minimized, HWND hWndOther)
EV_WM_ACTIVATEAPP	void EvActivateApp(bool active, HANDLE threadId)
EV_WM_ACTIVATEAPP	void EvActivateApp(bool active, HTASK hTask)
EV_WM_ASKCBFORMATNAME	void EvAskCBFormatName(uint bufLen, char far * buffer)
EV_WM_CANCELMODE	void EvCancelMode()
EV_WM_CHANGECBCHAIN	void EvChangeCBChain(uint bufLen, char far * buffer)
EV_WM_CHAR	void EvChar(uint key, uint repeatCount, uint flags)
EV_WM_CHARTOITEM	int EvCharToItem(uint key, HWND hWndListBox, uint caretPos)
EV_WM_CHILDACTIVATE	void EvChildActivate()
EV_WM_CHILDINVALID	void EvChildInvalid(HWND)
EV_WM_CLOSE	void EvClose()
EV_WM_COMPACTING	void EvCompacting(uint compactRatio)
EV_WM_COMPAREITEM	LRESULT EvCompareItem(uint ctrlId, COMPAREITEMSTRUCT ar&. compareInfo)
EV_WM_CREATE	int EvCreate(CREATESTRUCT far &.)
EV_WM_CTLCOLOR	HBRUSH EvCtlColor(HDC, HWND hWndChild, uint ctlType)
EV_WM_DEADCHAR	void EvDeadChar(uint deadKey, uint repeatCount, uint flags)
EV_WM_DELETEITEM	void EvDeleteItem(uint ctrlId, DELETEITEMSTRUCT far&. eleteInfo)
EV_WM_DESTROY	void EvDestroy()

续表

宏	响应函数声明
EV_WM_DESTROYCLIPBOARD	void EvDestroyClipboard()
EV_WM_DEVMODECHANGE	void EvDevModeChange(char far * devMode)
EV_WM_DRAWCLIPBOARD	void EvDrawClipboard()
EV_WM_DRAWITEM	void EvDrawItem(uint ctrlId, DRAWITEMSTRUCT far& drawInfo)
EV_WM_DROPFILES	void EvDropFiles(TDropInfo dropInfo)
EV_WM_ENABLE	void EvEnable(bool enabled)
EV_WM_ENDSESSION	void EvEndSession(bool endSession)
EV_WM_ENTERIDLE	void EvEnterIdle(uint source, HWND hWndDlg)
EV_WM_ERASEBKGND	bool EvEraseBkgnd(HDC)
EV_WM_FONTCHANGE	void EvFontChange()
EV_WM_GETDLGCODE	uint EvGetDlgCode(MSG far *)
EV_WM_GETFONT	HFONT EvGetFont();
EV_WM_GETMINMAXINFO	void EvGetMinMaxInfo(MINMAXINFO far &)
EV_WM_GETTEXT	void EvGetText(uint bufLen, char far * buffer)
EV_WM_GETTEXTLENGTH	uint EvGetTextLength() (WIN32 only)
EV_WM_HOTKEY	void EvHotKey(int idHotKey)
EV_WM_HSCROLL	void EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl)
EV_WM_HSCROLLCLIPBOARD	void EvHScrollClipboard(HWND hWndCBViewer, uint crollCode, uint pos)
EV_WM_ICONERASEBKGND	void EvIconEraseBkgnd(HDC)
EV_WM_INITMENU	void EvInitMenu(HMENU)
EV_WM_INITMENUPOPUP	void EvInitMenuPopup(HMENU hPopupMenu, uint index, bool ysMenu)
EV_WM_INPUTFOCUS	void EvInputFocus(bool gainingFocus) (WIN32 only)
EV_WM_KEYDOWN	void EvKeyDown(uint key, uint repeatCount, uint flags)
EV_WM_KEYUP	void EvKeyUp(uint key, uint repeatCount, uint flags)
EV_WM_KILLFOCUS	void EvKillFocus(HWND hWndGetFocus)
EV_WM_LBUTTONDBLCLK	void EvLButtonDblClk(uint modKeys, TPoint& point)
EV_WM_LBUTTONDOWN	void EvLButtonDown(uint modKeys, TPoint& point)
EV_WM_LBUTTONUP	void EvLButtonUp(uint modKeys, TPoint& point)
EV_WM_MBUTTONDBLCLK	void EvMButtonDblClk(uint modKeys, TPoint& point)
EV_WM_MBUTTONDOWN	void EvMButtonDown(uint modKeys, TPoint& point)
EV_WM_MBUTTONUP	void EvMButtonUp(uint modKeys, TPoint& point)
EV_WM_MDIACTIVATE	void EvMDIActivate(HWND hWndActivated, HWND WndDeactivated)
EV_WM_MDICREATE	LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct)
EV_WM_MDIDESTROY	void EvMDIDestroy(HWND hWnd)
EV_WM_MENUCHAR	uint EvMenuChar(uint nChar, uint menuType, HMENU hMenu)

续表

宏	响应函数声明
EV_WM_MENUSELECT	void EvMenuSelect(uint menuItemId, uint flags, HMENU hMenu)
EV_WM_MEASUREITEM	void EvMeasureItem(uint ctrlId, MEASUREITEMSTRUCT far& easureInfo)
EV_WM_MOUSEACTIVATE	uint EvMouseActivate(HWND hWndTopLevel, uint hitTestCode, int msg)
EV_WM_MOUSEMOVE	void EvMouseMove(uint modKeys, TPoint& point)
EV_WM_MOVE	void EvMove(TPoint &clientOrigin)
EV_WM_NCACTIVATE	bool EvNCActivate(bool active)
EV_WM_NCCALCSIZE	uint EvNCCalcSize(bool calcValidRects, NCCALCSIZE_PARAMS far &)
EV_WM_NCCREATE	bool EvNCCreate(CREATESTRUCT far &)
EV_WM_NCDESTROY	void EvNCDestroy()
EV_WM_NCHITTEST	uint EvNCHitTest(TPoint& point)
EV_WM_NCLBUTTONDBLCLK	void EvNCLButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCLBUTTONDOWN	void EvNCLButtonDown(uint hitTest, TPoint& point)
EV_WM_NCLBUTTONUP	void EvNCLButtonUp(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONDBLCLK	void EvNCMButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONDOWN	void EvNCMButtonDown(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONUP	void EvNCMButtonUp(uint hitTest, TPoint& point)
EV_WM_NCMOUSEMOVE	void EvNCMouseMove(uint hitTest, TPoint& point)
EV_WM_NCPAINT	void EvNCPaint()
EV_WM_NCRBUTTONDBLCLK	void EvNCRButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCRBUTTONDOWN	void void EvNCRButtonDown(uint hitTest, TPoint& point)
EV_WM_NCRBUTTONUP	void EvNCRButtonUp(uint hitTest, TPoint& point)
EV_WM_OTHERWINDOWCREATED	void EvOtherWindowCreated(HWND hWndOther)
EV_WM_OTHERWINDOWDESTROYED	void EvOtherWindowDestroyed(HWND hWndOther)
EV_WM_PAINT	void EvPaint()
EV_WM_PAINTCLIPBOARD	void EvPaintClipboard(HWND, HANDLE hPaintStruct)
EV_WM_PAINTICON	void EvPaintIcon()
EV_WM_PALETTECHANGED	void EvPaletteChanged(HWND hWndPalChg)
EV_WM_PALETTEISCHANGING	void EvPaletteIsChanging(HWND hWndPalChg)
EV_WM_PARENTNOTIFY	void EvParentNotify(uint event, uint childHandleOrX, uint childIDOrY)
EV_WM_POWER	int EvPower(uint powerEvent)
EV_WM_QUERYDRAGICON	HANDLE EvQueryDragIcon()
EV_WM_QUERYENDSESSION	bool EvQueryEndSession()
EV_WM_QUERYNEWPALETTE	bool EvQueryNewPalette()
EV_WM_QUERYOPEN	bool EvQueryOpen()
EV_WM_RBUTTONDBLCLK	void EvRButtonDblClk(uint modKeys, TPoint& point)

续表

宏	响应函数声明
EV_WM_RBUTTONDOWN	void EvRButtonDown(uint modKeys, TPoint& point)
EV_WM_RBUTTONUP	void EvRButtonUp(uint modKeys, TPoint& point)
EV_WM_RENDERALLFORMATS	void EvRenderAllFormats()
EV_WM_RENDERFORMAT	void EvRenderFormat(uint dataFormat)
EV_WM_SETCURSOR	bool EvSetCursor(HWND hWndCursor, uint hitTest, uint mouseMsg)
EV_WM_SETFOCUS	void EvSetFocus(HWND hWndLostFocus)
EV_WM_SETFONT	void EvSetFont(HFONT hFont, bool redraw)
EV_WM_SETTEXT	void EvSetText(char far * text)
EV_WM_SHOWWINDOW	void EvShowWindow(bool show, uint status)
EV_WM_SIZE	void EvSize(uint sizeType, TSize& size)
EV_WM_SIZECLIPBOARD	void EvSizeClipboard(HWND hWndViewer, HANDLE hRect)
EV_WM_SPOOLERSTATUS	void EvSpoolerStatus(uint jobStatus, uint jobsLeft)
EV_WM_SYSCHAR	void EvSysChar(uint key, uint repeatCount, uint flags)
EV_WM_SYSCOLORCHANGE	void EvSysColorChange()
EV_WM_SYSCOMMAND	void EvSysCommand(uint cmdType, TPoint& point)
EV_WM_SYSDEADCHAR	void EvSysDeadChar(uint key, uint repeatCount, uint flags)
EV_WM_SYSKEYDOWN	void EvSysKeyDown(uint key, uint repeatCount, uint flags)
EV_WM_SYSKEYUP	void EvSysKeyUp(uint key, uint repeatCount, uint flags)
EV_WM_SYSTEMERROR	void EvSystemError(uint error)
EV_WM_TIMECHANGE	void EvTimeChange()
EV_WM_TIMER	void EvTimer(uint timerId)
EV_WM_VKEYTOITEM	int EvVKeyToItem(uint key, HWND hWndListBox, uint caretPos)
EV_WM_VSCROLL	void EvVScroll(uint scrollCode, uint thumbPos, HWND hWndCtl)
EV_WM_VSCROLLCLIPBOARD	void EvVScrollClipboard(HWND hWndCBViewer, uint scrollCode, uint pos)
EV_WM_WINDOWPOSCHANGED	void EvWindowPosChanged(WINDOWPOS far &windowPos)
EV_WM_WINDOWPOSCHANGING	void EvWindowPosChanging(WINDOWPOS far &windowPos)
EV_WM_WININICHANGE	void EvWinIniChange(char far * section)

VBX 控件消息

表 B-12 所列的宏响应 WM_VBXFIREEVENT 消息。

表 B-12

宏	宏参数	响应函数声明
EV_VBXEVENTINDEX	ID, event, EvHandler	void EvHandler(VBXEVENT FAR * event)
EV_VBXEVENTNAME	ID, event, EvHandler	void EvHandler(VBXEVENT FAR * event)

用户自定义消息

表 B-13 的宏响应用户自定义消息。

表 B-13

宏	宏参数	响应函数声明
EV_MESSAGE	UserMessage, UserName	LRESULT UserName(WPARAM,LPARAM)
EV_REGISTERED	Registered name, UserName	LRESULT UserName(WPARAM,LPARAM)

附录 C

OWL 中的常数定义

C.1　BF_xxxx 按钮标志常数

头文件：checkbox.h

描述：检查框和 radio 按钮对象使用这个常数表明按钮的选择状态，见表 C-1。

表 C-1

常　　数	意　　义
BF_CHECKED	选择状态
BF_GRAYED	灰色状态
BF_UNCHECKED	非选择状态

C.2　CM_xxxx 编辑命令常数

头文件：window.rh

描述：编辑菜单项的菜单选择命令，见表 C-2。

表 C-2

常　　数	成员函数	对应的菜单项
CM_EDITCLEAR	TEdit::CMEditClear	Edit|Clear
CM_EDITCOPY	TEdit::CMEditCopy	Edit|Copy
CM_EDITCUT	TEdit::CMEditCut	Edit|Cut
CM_EDITDELETE	TEdit::CMEditDelete	Edit|Delete
CM_EDITPASTE	TEdit::CMEditPaste	Edit|Paste
CM_EDITUNDO	TEdit::CMEditUndo	Edit|Undo
CM_EXIT	TWindow::CmExit	File|Exit

C.3 CM_xxxx 编辑文件命令常数

头文件：docview.rh

描述：编辑文件的菜单项的命令，见表 C-3。

表 C-3

常　数	成员函数	对应的菜单项
CM_FILECLOSE		File|Close
CM_FILENEW	TEditFile::CmFileNew	File|New
CM_FILEOPEN	TEditFile::CmFileOpen	File|Open
CM_FILEPRINT		File|Print
CM_FILEPRINTERSETUP		File|PrinterSetup
CM_FILEREVERT	TDocManager::CmFileRevert	File|Revert
CM_FILESAVE	TEditFile::CmFileSave	File|Save
CM_FILESAVEAS	TEditFile::CmFileSaveAs	File|Save As
CM_VIEWCREATE	TDocManager::CmViewCreate	File|View Create

C.4 CM_xxxx 编辑替换命令常数

头文件：editsear.rh

描述：编辑替换的菜单项命令，见表 C-4。

表 C-4

常　数	成员函数	对应的菜单项
CM_EDITFIND	TEditSearch::CMEditFind	Edit|Find
CM_EDITFINDNEXT	TEditSearch::CMEditFindNext	Edit|Find|Next
CM_EDITREPLACE	TEditSearch::CMEditReplace	Edit|Replace

C.5 CM_xxxx 多文档界面命令常数

头文件：mdi.rh

描述：多文档界面的菜单项选择命令，见表 C-5。

表 C－5

常　　数	成员函数	对应的菜单项
CM_ARRANGEICONS	TMDIClient::CmArrangeIcons	Window | Arrange Icons
CM_CASCADECHILDREN	TMDIClient::CmCascadeChildren	Window | Cascade
CM_CLOSECHILDREN	TMDIClient::CmCloseChildren	Window | Close All
CM_CREATECHILD	TMDIClient::CmCreateChild	
CM_TILECHILDREN	TMDIClient::CmTileChildren	Window | Tile
CM_TILECHILDRENHORIZ	TMDIClient::CmTileChildren	Window | Tile

C.6　dmxxxx 文档管理器模式常数

头文件：docmanag.h

描述：被 TDocManager 类用于表明一个文档的属性，见表 C－6。

表 C－6

常　　数	意　　义
dmMenu	为文件菜单设置 ID 值
dmMDI	提供多个打开的文档
dmNoRevert	关闭菜单项 File | Revert 的功能
dmSaveEnable	开启菜单项 File | Save 的功能
dmSDI	不提供多个打开的文档

C.7　dtxxxx 文档样板常数

头文件：locale.h

描述：被 TDocument 和 TDocTemplate 使用创建样板，见表 C－7。

表 C-7

常　数	Windows 中的等价常数	意　　义
dtAutoDelete		当最后一个视口删除时删除文档
dtAutoOpen		自动打开一个文档
dtCreatePrompt	(OFN_CREATEPROMPT)	如果文档不存在，创建文档是就警告用户
dtDynRegInfo		用于 OLE 2 应用程序
dtFileMustExist	(OFN_FILEMUSTEXIST)	让用户只能输入已存在的文件名，如果输入错误就发出警告
dtHidden		隐藏用户选择样板
dtHideReadOnly	(OFN_HIDEREADONLY)	隐藏只读检查框
dtNewDoc		创建一个新的文档，但不定义路径
dtNoAutoView		不自动创建缺省的文件类型
dtNoReadOnly	(OFN_NOREADONLYRETURN)	返回可写的特定文件
dtNoTestCreate	(OFN_NOTESTFILECREATE)	不进行文件创建时的检查
dtOverwritePrompt	(OFN_OVERWRITEPROMPT)	当 Save As 对话框显示时提问用户是否覆盖文件
dtPathMustExist	(OFN_PATHMUSTEXIST)	只允许有效的文件路径输入，否则就显示警告信息
dtProhibited	(OFN_ALLOWMULTISELECT) (OFN_ENABLEHOOK) (OFN_ENABLETEMPLATE) (OFN_ENABLETEMPLATEHANDLE)	不提供这些特定的窗口选择
dtReadOnly	(OFN_READONLY)	当对话框创建时显示只读检查框
dtRegisterExt		用于 OLE 2 应用程序
dtSelected		表明最后选择的样板
dtSingleUse		表明文档被注册为单用户文档
dtSingleView		给每一个文档只提供一个视口
dtUpdateDir		根据对话框中特定的目录更新目录

C.8　ID_xxxx 文件常数

头文件：inputdia. rh

描述：输入对话框的资源和控件 ID 值，见表 C-8。

表 C-8

常　数	意　　义
IDD_INPUTDIALOG	输入对话框的资源 ID 值
ID_INPUT	用户输入的控件 ID 值
ID_PROMPT	静态控件的 ID 值

C.9　ID_xxxx 打印常数

头文件：printer.rh

描述：打印退出对话框的资源和控件 ID 值，见表 C-9。

表 C-9

常　数	意　义
IDD_ABORTDIALOG	打印退出对话框的资源 ID 值
ID_TITLE	选择打印机驱动器的控件 ID 值
ID_DEVICE	所选择打印机的控件 ID 值
ID_PAGE	页码文本控件的控件 ID 值
ID_PORT	所选择的打印端口控件 ID 值

C.10　IDA_EDITFILE 加速键 ID 常数

头文件：editfile.rh

常数：IDA_EDITFILE

描述：加速键的资源 ID 值。

C.11　IDA_OLEVIEW OLE 加速键 ID 常数

头文件：oleview.rh

常数：IDA_OLEVIEW

描述：OLE 应用程序中的加速键的资源 ID 值。

C.12　IDM_EDITFILE 菜单 ID 常数

头文件：editfile.rh

常数：IDM_EDITFILE

描述：菜单选择的资源 ID 值。

C.13　IDS_xxxx 文档字符串 ID 常数

头文件：docview.rh

描述：定义资源 ID 值的字符串 ID 值，用来确定文档的状态，见表 C-10。

表 C-10

常　数	显示信息
IDS_DOCCHANGED	Do you want to save the changes?
IDS_DOCLIST	Document is a document type.
IDS_DOCMANAGERFILE	This is a document manager file.
IDS_DUPLICATEDOC	This is a duplicate document.
IDS_NODOCMANAGER	There is no document manager.
IDS_NOMEMORYFORVIEW	Not enough memory to view the document
IDS_NOTCHANGED	The document has not been changed.
IDS_READERROR	Error while reading the file
IDS_UNTITLED	The file is untitled.
IDS_UNABLECLOSE	Document manager is unable to close the document.
IDS_UNABLEOPEN	Document manager is unable to open the document.
IDS_UNTITLED	Document is untitled.
IDS_VIEWLIST	Document is a view type.

C.14　IDS_xxxx 编辑文件 ID 常数

头文件：editfile.rh

描述：文件的编辑信息字符串，见表 C-11。

表 C-11

常　数	显示字符串
IDS_FILECHANGED	The text in the file has changed. Do you want to save the changes?
IDS_FILEFILTER	Use this filter to display text files.
IDS_UNABLEREAD	Unable to read the file from the disk.
IDS_UNABLEWRITE	Unable to write the file to the disk.
IDS_UNTITLEDFILE	The default window title unless the file is being edited.

C.15　IDS_xxxx 异常管理常数

头文件：except.rh

描述：通常的和应用程序异常的消息常数，见表 C-12。

表C-12

常 数	意 义
IDS_INVALIDMAINWINDOW	无效主窗口
IDS_INVALIDMODULE	无效的窗口模块
IDS_NOAPP	没有应用程序对象
IDS_OKTORESUME	错误恢复
IDS_OWLEXCEPTION	未知的异常
IDS_OUTOFMEMORY	超出内存范围
IDS_UNHANDLEDXMSG	不能除了错误
IDS_UNKNOWNERROR	未知的错误
IDS_UNKNOWNEXCEPTION	未知的异常错误
(以下为Owl I 兼容信息)	
IDS_INVALIDCHILDWINDOW	无效主窗口
IDS_INVALIDCLIENTWINDOW	无效客户窗口
IDS_INVALIDWINDOW	无效的窗口
(以下为TXWindow 信息)	
IDS_CHILDCREATEFAIL	子窗口创建失败
IDS_CHILDREGISTERFAIL	子窗口注册失败
IDS_CLASSREGISTERFAIL	类型注册失败
IDS_LAYOUTCOMPLETE	Layout 窗口失败
IDS_LAYOUTBADRELWIN	Layout 窗口失败
IDS_MENUFAILURE	菜单创建失败
IDS_PRINTERERROR	打印机错误
IDS_VALIDATORSYNTAX	语法确认失败
IDS_WINDOWCREATEFAIL	窗口创建失败
IDS_WINDOWEXECUTEFAIL	窗口执行失败
(以下为GDI 消息)	
IDS_GDIALLOCFAIL	GDI 部署失败
IDS_GDICREATEFAIL	GDI 创建失败
IDS_GDIDELETEFAIL	GDI 对象删除失败
IDS_GDIDESTROYFAIL	GDI 对象破坏失败
IDS_GDIFAILURE	GDI 错误
IDS_GDIFILEREADFAIL	GDI 读文件失败
IDS_INVALIDDIBHANDLE	无效的 DIB 句柄
IDS_GDIRESLOADFAIL	GDI 资源装载失败

C.16　IDS_xxxx 列表显示 ID 常数

头文件：listview. rh

常数：IDS_LISTVIEW

描述：列表显示的资源 ID 常数。

C.17　IDS_xxxx 打印字符串 ID 常数

头文件：printer. rh

描述：用来确定打印状态的字符串常数，见表 C-13。

表 C-13

常　数	显示字符串
IDS_PRNCANCEL	撤销打印
IDS_PRNERRORCAPTION	发生打印错误
IDS_PRNERRORTEMPLATE	文件没有打印
IDS_PRNGENERROR	打印中发生错误
IDS_PRNMGRABORT	从打印管理器中清除打印任务
IDS_PRNON	打印机
IDS_PRNOUTOFDISK	磁盘空间不够
IDS_PRNOUTOFMEMORY	内存不够

C.18　IDS_xxxx validatorID 常数

头文件：validate. rh

描述：定义类使用的常数，确定 validator 的状态，见表 C-14。

表 C-14

常　数	意　义
IDS_VALPXPCONFORM	项目不符合正确的图片格式
IDS_VALINVALIDCHAR	字符不是有效的入口
IDS_VALNOTINRANGE	入口没有特定的范围
IDS_VALNOTINLIS	在有效的入口中没有找到字符串

C.19　IDW_MDICLIENT 常数

头文件：framewin.h
常数：IDW_MDICLIENT
描述：定义 MDI 客户窗口的子窗口 ID 常数。

C.20　IDW_FIRSTMDICHILD 常数

头文件：framewin.h
常数：IDW_FIRSTMDICHILD
描述：定义 MDI 的第一个客户窗口的子窗口 ID 常数。

C.21　lmParent 常数

头文件：layoutco.h
定义：#define lmParent 0
描述：lmParent 常数用作排列。

C.22　MAX_RSRC_ERROR_STRING 常数

头文件：except.h
定义：const int MAX_RSRC_ERROR_STRING = 255;
描述：错误信息的最大字符数目。

C.23　pfxxxx 性能属性常数

头文件：docview.h
描述：定义文档和浏览性能属性，见表 C-15。、

表 C-15

常　数	意　义
pfGetText	可以访问文本格式
pfGetBinary	可以访问特殊的发文本格式
pfConstant	对象创建后属性就不再改变
pfSettable	属性可以重新设置
pfUnknown	定义属性但对象不能使用
pfHidden	属性隐藏起来
pfUserDef	在运行时由用户定义属性

C. 24　vnxxxx 视口通知常数

头文件：docview. h

描述：视口通知常数用来通知一个事件，见表 C-16。

表 C-16

常　　数	意　　　义
vnCommit	改变影响了文档
vnCustomBase	文档通知的基本事件
vnDocOpened	文档已经打开
vnDocClosed	文档已经改变
vnIsDirty	如果是非影响的改变存在，值为 true
vnInvalidate	如果是非影响的改变存在，值为 true
vnIsWindow	如果 HWND 使用这个视口，值为 true
vnRevert	文档原来的数据被装载代替当前的数据
vnViewOpened	构造了一个新的视口
vnViewClosed	一个视口关闭了

C. 25　TValidatorOptions 联合

头文件：validate. h

定义：enum TValidatorOptions {voFill, voTransfer, voOnAppend, voReserved };

描述：代表位图设置选择，见表 C-17。

表 C-17

常　　数	意　　　义
voFill	是否填充一个字符
voTransfer	传输输入行数据
voOnAppend	任何除了对话框控件
voReserved	保存一个位

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):
{
"filename": "XzEwOTQwMTMxLnppcA==",
"filename_decoded": "_10940131.zip",
"filesize": 51031619,
"md5": "09dac6bfd7b1232eeaf0683f2133b1fa",
"header_md5": "4951c5f4f848cd7dcfb4cb0c3cb8b0dc",
"sha1": "d1142ab53613b99dc6bd5bdc6624f7ccdef3c4fe",
"sha256": "7b0fc0620377fc8cdd840104e15ebce5c728b2a9939ec543ddae98f1b8d41f75",
"crc32": 715453732,
"zip_password": "",
"uncompressed_size": 56942361,
"pdg_dir_name": "",
"pdg_main_pages_found": 424,
"pdg_main_pages_max": 424,
"total_pages": 432,
"total_pixels": 3036396366,
"pdf_generation_missing_pages": false
}